Environmental timeseries lifecycle in the Internet of Things era

Lowering e-science barriers

Argyrios Samourkasidis

Environmental timeseries lifecycle in the Internet of Things era

Lowering e-science barriers

Argyrios Samourkasidis

Environmental timeseries lifecycle in the Internet of Things era

Lowering e-science barriers

Argyrios Samourkasidis

Thesis
submitted in fulfilment of the requirements for the degree of doctor
at Wageningen University
by the authority of the Rector Magnificus
Prof. Dr A.P.J. Mol,
in the presence of the
Thesis Committee appointed by the Academic Board
to be defended in public
on Monday 14th of October 2019
at 11 a.m. in the Aula.

Argyrios Samourkasidis
Environmental timeseries lifecycle in the Internet of Things era
Lowering e-science barriers
126 pages.

PhD thesis, Wageningen University, Wageningen, NL (2019)
With references, and summary in English

ISBN 978-94-6395-070-1
DOI 10.18174/497755

List of Abbreviations

Acronym	Definition
AGPL	Affero General Public License
API	Application Programming Interface
APSIM	Agricultural Production Systems sIMulator
ARM	Advanced RISC Machine
ART	Average Response Time
AgMIP	Agricultural Model Intercomparison and Improvement Project
BD	Big Data
BoM	Bureau of Meteorology
CBM	Community Based Monitoring
CO	Carbon Monoxide
CO-OPS	Center for Operational Oceanographic Products and Services
CPU	Central Processing Unit
CSV	Comma Separated Values
CUAHSI	Consortium of Universities for the Advancement of Hydrologic Science
CoAP	Constrained Application Protocol
DAC	Divide And Conquer (algorithm)
DDR	Double Data Rate
DSSAT	Decision Support System for Agrotechnology Transfer
EDAM	Environmental Data Acquisition Module
EEA	European Environmental Agency
EIoT	Environmental Internet of Things
EPA	Environmental Protection Agency
FAIR	Findable Accessible Interoperable Reusable
GB	Giga Byte
GHz	Giga Hertz
GMT	Greenwich Mean Time
GNU	GNU's not Unix
GPIO	General Purpose Input Output
GPS	Global Positioning System
GPU	Graphics Processing Unit

Acronym	Definition
GUI	Graphical User Interface
HAT	Hardware Attached on Top
HIS	Hydrologic Information System
HOA	Hydrological Observatory of Athens
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
IoT	Internet of Things
JSON	JavaScript Object Notation
KNMI	Koninklijk Nederlands Meteorologisch Instituut
LEC	LEaky Client
LED	Light Emitting Diode
MB	Mega Byte
MHz	Mega Hertz
NOAA	National Oceanic and Atmospheric Administration
OAI	Open Archive Initiative
ODM	Observations Data Model
OGC	Open Geospatial Consortium
O&M	Observations and Measurements
ORM	Object Relational Mapping
OSS	Open Source Solutions
OWL	Web Ontology Language
PAC	Pagination Aware Client
PMH	Protocol Metadata Harvesting
QA	Quality Assurance
QC	Quality control
RAM	Random Access Memory
REST	REpresentational State Transfer
RH	Relative Humidity
RPS	Requests Per Second
RQ	Research Question
RTC	Real Time Clock
SD	Secure Digital
SI	International System (of Units)
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOS	Sensor Observation Service
SPARQL	SPARQL Protocol And RDF (Resource Description Framework) Query Language
SPI	Serial Peripheral Interface
SQL	Structured Query Language

Acronym	Definition
SSH	Secure Shell
SWE	Sensor Web Enablement
SWEET	Semantic Web for Earth and Environment Technology
TPH	Tropical and Public Health Institute
TSV	Tab Separated Values
UAV	Unmanned Aerial Vehicle
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
VGI	Volunteered Geographic Information
WFS	Web Feature Service
WMO	World Meteorological Organization
WOFOST	WOrld FOod STudies
WS	Web Services
WSN	Wireless Sensor Network
XML	eXtensible Markup Language

Contents

Chapter 1

Introduction

This chapter is based on:

Samourkasidis, A., Athanasiadis, I. N. Environmental timeseries lifecycle in the IoT era. In: M. Arabi, O. David, J. Carlson, D. P. Ames (Eds.), Proceedings of the 9th Intl. Congress on Environmental Modelling and Software [Manuscript accepted for publication]

1.1 Problem statement

One key objective of the environmental data science is to narrow the data-to-knowledge latency (Elag et al., 2017). According to Foster et al. (2012), *scientific data doubles every twelve months, which is often faster than it can be converted into useful knowledge.* Knowledge is produced when raw data are processed through scientific tools, as environmental models. Domain scientists curate relevant datasets, feed them as input to scientific tools, and interpret tool outputs into actionable knowledge (Gibert et al., 2018; Athanasiadis & Mitkas, 2007). Nowadays, e-scientists and environmental practitioners are confronted with the ever-growing amount of environmental datasets, and new data produced by the Internet of Things (IoT). Not only raw data have to undergo certain modifications in order to be processed by environmental models (Rizzoli et al., 2007), but also new computing requirements are introduced, in order this vast amount of data to be processed (Good et al., 2017).

The Internet of Things (IoT), that is the "ensemble of applications and services" of interconnected smart objects equipped with sensors (Miorandi et al., 2012), could support e-scientists in two ways. Firstly, some IoT components (i.e. IoT gateways (Kruger & Hancke, 2014)) could facilitate environmental data storage and dissemination, enabling e-scientists to deploy low-cost, yet reliable, environmental monitoring campaigns. Secondly, the IoT ecosystem can provide e-scientists with spatially and temporally diverse environmental datasets. This is why, the rise of IoT boosted the citizen-science movement (Silvertown, 2009). In the literature, citizen-science is expressed through the concepts of Community-Based Monitoring (CBM) and Volunteered Geographic Information (VGI) (Conrad & Hilchey, 2011; Connors et al., 2012). These concepts are often realized as IoT-enabled environmental campaigns: IoT devices are deployed and collect environmental timeseries (Santos et al., 2018; Sanchez et al., 2014).

Despite the aforementioned opportunities, the IoT ecosystem brings new challenges to the environmental data lifecycle. Traditional environmental data lifecycle mainly concerns data storage, dissemination, acquisition and integration (Athanasiadis & Mitkas, 2004; Mason et al., 2014). The restrained capabilities of the IoT ecosystem (Atzori et al., 2010) questions traditional methodologies about data storage and dissemination. Heterogeneity, syntactic or semantic, pertinent to the IoT ecosystem challenges environmental data acquisition and integration (Horsburgh et al., 2009). An assessment of the impact of the IoT stressors on the environmental data lifecycle is needed, in order to reap the benefits from the great variety of spatially and temporally diverse environmental datasets that IoT offers.

The IoT prototyping devices can play a more central role in the environmental data lifecycle. These devices, such as Raspberry Pi[1], are being increasingly utilized in environmental monitoring campaigns (Cagnetti et al., 2013; Leccese et al., 2014; Samourkasidis & Athanasiadis, 2014; Nikhade, 2015). Despite their low acquisition cost and their multi-purpose nature, their use is limited to auxiliary operations. In most cases, data storage and processing takes place in other nodes (Moure et al., 2015). The weak enabling environment (i.e. opportunistic Internet/power connection) under which they operate, questions their capabilities for persistent data storage.

[1]https://www.raspberrypi.org/about/

Besides data storage, the weak IoT enabling environment challenges the operation of environmental data management frameworks. These frameworks, such as the Sensor Observation Service of the Open Geospatial Consortium (OGC), provide standardized ways for timeseries data discovery and access, accounting for syntactic interoperability (Bröring et al., 2012). Environmental data management frameworks facilitate e-scientists to extract and exchange data and interact with scientific workflows by providing inputs to environmental models (Regueiro et al., 2015). However, there are certain contraints, mostly in terms of performance and efficiency, in order for these frameworks to operate on IoT devices (Jirka et al., 2012; Pradilla et al., 2015; Jazayeri et al., 2015).

The IoT ever increasing syntactic and semantic heterogeneity obstructs e-scientists towards data acquisition and integration. To date, environmental data acquisition and integration are the most time consuming processes of the lifecycle (Horsburgh et al., 2016). In principle, e-scientists have to a) obtain datasets available through various dissemination protocols, and formatted under custom data syntaxes, b) transform them into a common data format and c) feed them as input into scientific tools (i.e. environmental models), in order to transform raw data into actionable knowledge. IoT will only exacerbate this process, as new data formats/syntaxes emerge constantly (Atzori et al., 2010). Syntactic and semantic heterogeneity, that is the difference among the syntaxes, definitions and representations of the involved datasets, render the aforementioned workflow into a highly custom and manual process, as acquisition, transformation and integration into common data formats requires almost always manual involvement from experts (Horsburgh et al., 2009; Mason et al., 2014; Samourkasidis et al., 2018). In the context of environmental data literature, there are various analytical methods to cope with syntactic and semantic heterogeneity. These include, but not limited to, acquisition facilitated by environmental data management frameworks (Bröring et al., 2011a; Horsburgh et al., 2009); scripting (Woodard, 2016; Porter et al., 2014); and integration supported by Semantic Web technologies and ontology engineering (Ziébelin et al., 2017).

1.2 Research objective

The objective of this thesis is to *assess the impact of IoT on environmental timeseries data lifecycle from the perspective of e-scientists*. We identify two IoT stressors, the IoT ecosystem *restrained resources* and the *syntactic and semantic heterogeneity*, and investigate their impact on the environmental timeseries lifecycle processes. We further refined the main objective into three sub-objectives. The first two concern the evaluation of environmental timeseries data storage and dissemination under the light of the IoT ecosystem restrained resources. The last sub-objective concerns the impact of IoT *heterogeneity*, syntactic and semantic, on environmental timeseries acquisition and integration, respectively. The sub-objectives, formulated as Research Questions (RQs) are the followings:

a. RQ1: Can environmental timeseries lifecycle be facilitated by IoT prototyping devices?
b. RQ2: Are environmental data dissemination protocols IoT-ready?
c. RQ3: How can e-scientists acquire, integrate and transform environmental timeseries datasets in the heterogeneous IoT ecosystem?

1.3 Background and related work

In this section, we set the context by defining key concepts of this thesis, as IoT gateways, environmental timeseries and e-scientists. First, we present our perspective on Internet of Things, its associated building blocks and identify the main challenges. Then, we present our view on environmental timeseries lifecycle, describe the role of e-scientists within this lifecycle, and identify the main barriers. Figure 1.1 depicts how the involved components of this thesis interact with each other: IoT devices are deployed in order to collect environmental datasets (Santos et al., 2018; Sanchez et al., 2014). An IoT gateway stores the collected data and facilitates their dissemination, so they can be reused and further processed. As a result, more spatially and temporally diverse environmental datasets are available and e-scientists can use them along with traditional data sources, in order to complement their research. These datasets may be a) disseminated through various protocols, b) stored under different syntaxes and c) annotated with custom semantics. Thus, e-scientists firstly acquire these datasets and then transform and integrate them into one format in order to use them as input in their scientific tools (e.g. environmental models). The lifecycle continues, as the the output of these models can be stored and disseminated for further processing.

1.3.1 Internet of Things and challenges

Internet of Things comprises of interconnected devices, which sense their surrounding environment and report observations on the web (Gubbi et al., 2013; Atzori et al., 2010). The Internet of Things is structured in three main Layers: Perception, Network, Application (Mahmoud et al., 2015). The Perception (or Device) layer comprises of the IoT devices, which are digitized devices equipped with sensors. The Network (or Transmission) layer handles the data transmission from the previous layer to IoT gateways. An *IoT prototyping device* or *IoT gateway* serves as an intermediate among the IoT devices and the Internet (Kruger & Hancke, 2014). The Application is the highest layer in the IoT architecture and serves as the interface between the Internet and the IoT devices which are part of a specific domain (e.g. smart city, smart agriculture, etc.). According to Khan et al. (2012) there are two additional layers: a) the Middleware, which follows Network and precedes Application layer, and b) the Business layer which stands at the top of the IoT architecture. Over time, Middleware and Business layers were incorporated into the Application layer. In this thesis, we take a high-level view on the IoT architecture and focus on IoT challenges which are derived by the architecture and the attributes of its components.

The IoT challenges are relevant in the environmental timeseries lifecycle. Some of the IoT challenges are: a) the restrained resources ecosystem, b) heterogeneity, c) privacy and security, d) lack of standardization (Atzori et al., 2010; Chiang & Zhang, 2016). In this thesis we focus on a) and b). The restrained resources ecosystem refer to the limited capabilities of IoT devices in terms of processing power and network bandwidth (Atzori et al., 2010; Chiang & Zhang, 2016). These limited capabilities, can have an impact when it comes to persistent data storage, as well as efficient and interoperable data dissemination. Regarding this challenge, we focus on IoT prototyping devices which are connected to power supply and have access to

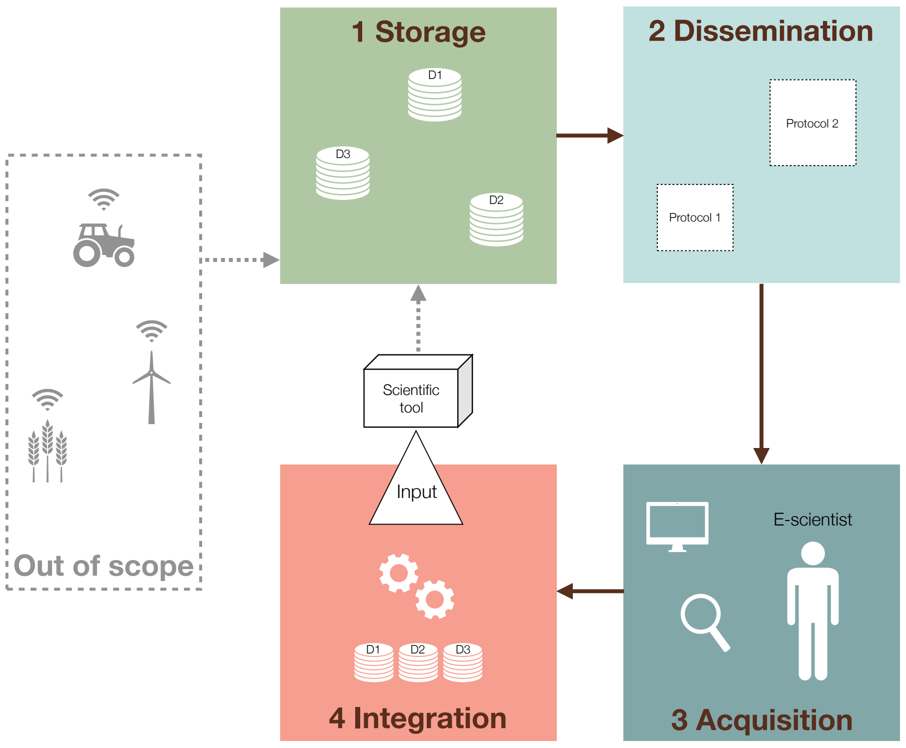


Figure 1.1: The interaction of this thesis components within an environmental lifecycle scenario. The outer left box, which is out of the scope of this thesis, depicts the various data sources. These can be IoT-produced datasets and/or the output of scientific tools such as environmental models. The different databases in Box **1** represent the different syntaxes and semantics under which the datasets are stored. Similarly, in Box **2** datasets can be disseminated via various dissemination protocols. Box **3** represents the process where e-scientists acquire datasets of interest. The cogwheels in Box **4** depict the integration methods which transform the acquired datasets into a certain syntax (Input dataset), so this can be used as input to a scientific tool. The output of the scientific tool can be stored, continuing the environmental data lifecycle.

the Internet. Heterogeneity, syntactic and semantic, is another key IoT challenge (Miorandi et al., 2012). The diverse custom formats used by IoT devices to report their readings hinder e-scientists to gain knowledge out of the large data volumes (Alansari et al., 2018; Chiang & Zhang, 2016). Human expert intervention is almost always required to extract insights from raw data when syntactic and semantic heterogeneous datasets are involved (Athanasiadis, 2015). In this thesis, we investigate the impact of both IoT-related challenges (restrained resources and heterogeneity) on the environmental timeseries lifecycle.

1.3.2 Environmental timeseries lifecycle and e-science

Environmental timeseries lifecycle is a set of processes which accompany data since their creation. Mason et al. (2014) describe the processes of storage, publication, acquisition, integration and others (e.g. visualization, quality assurance/quality control, etc.) as data management. That is "the task of shepherding data through the various components of the data lifecycle" (Mason et al., 2014). The environmental timeseries lifecycle processes can be viewed from the perspective of different user roles: From a *data producer* viewpoint, first comes the creation and the collection of the timeseries, then follows data storage and finally is the dissemination. For example, Horsburgh et al. (2011) present an environmental observatory information system, whose main components are among others the persistent data storage and interoperable publication of environmental timeseries. In the literature, the terms data publication (Horsburgh et al., 2009), dissemination (Beaujardière, 2016) and sharing (Langegger et al., 2008) are synonyms and used interchangeably. From a *data consumer* viewpoint, the main processes are the data acquisition and integration. In this context, we refer to acquisition as the process of making use of a dissemination channel in order to acquire data. In a sense, the dissemination and acquisition processes have an inverse relationship. By the term integration we refer to the process of combining (acquired) timeseries into a specific format in order to be further processed (Beran & Piasecki, 2009). In this thesis, we use the term *environmental timeseries lifecycle* to refer to the following steps: storage, dissemination, acquisition and integration.

Standardization of the environmental data lifecycle has been always in the spotlight, in order to enable scientific data reusability and reproducibility. Standardization has been investigated from a) abstract, b) process or c) holistic viewpoints.

Wilkinson et al. (2016) present the FAIR (Findable, Accessible, Interoperable, Reusable) Guiding Principles, which concern a set of abstract best-practices to support e-scientists and data producers towards discovering and reusing scientific data. Process-based standardization utilize one standard for each process. For example, the Observations Data Model (ODM) of the CUAHSI Community (Horsburgh et al., 2016) is concerned with storage, and the OGC Sensor Observation Service (Bröring et al., 2012) with the dissemination and acquisition process. Note that the data integration process is not directly supported by any standard. A variety of approaches can be used, including ontology engineering (Ziébelin et al., 2017), Linked Data (Harth et al., 2013), and mediator-wrapper architectures (Regueiro et al., 2015), among others. Finally, the holistic standardization is related to environmental cyberinfrastructures, i.e. systems which fulfil most or all processes of the environmental timeseries lifecycle (Horsburgh et al., 2009; Ames et al., 2012).

E-science is a broad term which usually concerns a data-intensive approach for hypothesis investigation. According to the definition given on the 2018 IEEE International Conference on eScience website: “eScience promotes innovation in collaborative, computationally- or data-intensive research across all disciplines, throughout the research lifecycle” (IEEE International Conference on eScience, 2018). There are several more definitions for e-science, and most of them feature the following attributes: a) cross-discipline collaboration among individuals, b) data-intensive approach to investigate research hypothesis, and c) a requirement for very large scale computing resources (Hey & Trefethen, 2005; Jankowski, 2007). In this thesis, the last attribute was less relevant.

1.3.3 IoT impact on environmental timeseries lifecycle

The IoT-related challenges have an impact on environmental timeseries lifecycle. Good et al. (2017) mention that e-scientists have to overcome two types of barriers: computing requirements and technological skills. The former is concerned with the need for more computing power in order to process the vast amount of IoT-produced data. The latter refers to the technical expertise required from the e-scientists in order to transform raw data into actionable knowledge. Computing aspects may affect the storage and dissemination steps of the environmental data lifecycle, and the technological skills affect the acquisition and integration ones. In this thesis, we focus on the a) computing requirements, by investigating how can IoT gateways support storage and dissemination, and b) technological skills, by investigating how declarative approaches can facilitate e-scientists towards acquiring and integrating environmental timeseries data.

In the environmental domain literature there are efforts to lower both barriers. Firstly, there is an ongoing trend to utilize more and more the computing power of the low-cost IoT gateways to support environmental data related purposes. Raspberry Pi is an IoT prototyping device, which is considered an “IoT enabler technology” (Johnston & Cox, 2017), and is used in diverse range of applications (e.g. smart city (Re et al., 2014; Jung et al., 2013; Leccese et al., 2014), smart home (Chowdhury et al., 2013; Bahrudin et al., 2013; Vujović & Maksimović, 2015), etc.). In the environmental monitoring context, Raspberry Pi may facilitate one, but not all of the followings simultaneously: data processing, (temporary) storage and dissemination of the observed data to another node (or the cloud) for further processing and persistent storage (Johnston & Cox, 2017; de Assis et al., 2016). For example, Moure et al. (2015) developed a low-cost, real-time volcanic activity monitor based on a Raspberry Pi, which enables users to retrieve data through a commercial texting application. The common attribute in all of the above examples is that Raspberry Pi has not been used for permanent data storage. In this thesis, we worked with the Raspberry Pi due to its wide adoption, open source nature, generic purpose, and low-acquisition cost.

The limited computing capabilities of IoT components affect also the environmental timeseries dissemination. The OGC Sensor Observation Service is a popular environmental timeseries dissemination protocol. In 2006 the Open Geospatial Consortium introduced SOS within the context of Sensor Web Enablement (Botts et al., 2008). SOS has been adopted by several environmental agencies, including EPA (Environmental Protection Agency, 2016) and NOAA/CO-OPS (Center for Operational Oceanographic Products and Services (CO-OPS), 2019), as it supports interoperable environmental timeseries dissemination. The European Environment Agency (EEA) in the context of the INSPIRE directive designed and implemented a framework according to which all member states report their environmental datasets through OGC SOS instances (Kjeld et al., 2011; Jirka et al., 2012; Jirka & Bröring, 2012). We selected to investigate OGC SOS, instead of the other OGC initiative which is intended for IoT, such as the OGC SensorThings API. This decision is based on the widespread adoption of OGC SOS in comparison with the limited one of the SensorThings API. While the latter was designed for IoT applications, it is still under development, and it will take a while until it is fully adopted by environmental agencies. Within the IoT context, there are a lot of efforts in order to utilize OGC SOS for data dissemination through IoT devices by proposing different low-level implementations (Jazayeri et al., 2015). For example, Jazayeri et al. (2012) introduced TinySOS, a lightweight OGC SOS implementation to be hosted on IoT devices. TinySOS makes use of an optimized web service and XML processing engine in order to compensate for the constrained capabilities of IoT devices. Similarly, Pradilla et al. (2015) propose SOSLite, another lightweight OGC SOS implementation by utilizing a NoSQL database. Finally, Pradilla et al. (2016) prototype on IoT Networking Layer by proposing SOS CoAP (Constrained Application Protocol), an OGC SOS version to operate on IoT. In this thesis, we focus on OGC SOS as an indicative example of standardized dissemination protocols.

The required technological skills is a barrier which hinders e-scientists to acquire and integrate heterogeneous environmental datasets in order to analyze them. Nowadays, an environmental e-scientist besides their domain expertise should have certain computer science skills in order to process, analyze and transform raw data into actionable knowledge (Gibert et al., 2018). In the environmental science literature, there is a number of efforts which utilize computer science skills in order to acquire and transform syntactically and semantically diverse environmental timeseries (Woodard, 2016; Porter et al., 2014; Stadtmüller et al., 2013; Harth et al., 2013). Swain et al. (2016) attempted to lower this barrier by a framework which allows environmental scientists to develop web applications writing a very small amount of code. In this thesis, we focus on declarative approaches to cope with syntactic and semantic heterogeneity. We argue that declarative approaches are fit-for-purpose for e-scientists without a very strong computer science background.

The aforementioned challenges that come with IoT, add up to the longstanding challenge of curating legacy environmental datasets. Besides the technical challenges that the new IoT devices introduce, there is also a semantic barrier that e-scientists have to overcome. This is because environmental sciences are fragmented and different semantics are used by different sub-disciplines, creating heterogeneity.

1.4 Methodology and thesis outline

This section presents the methodology we followed in order to *assess the impact of IoT on environmental timeseries lifecycle from the perspective of e-scientists.* Figure 1.2 depicts

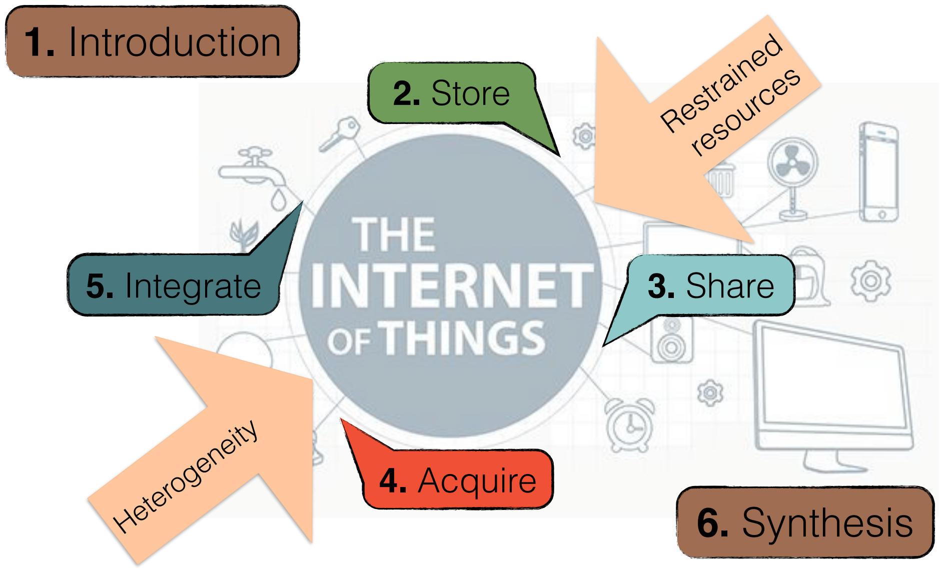


Figure 1.2: The overview of this thesis along with an illustration of the IoT stressors impact against the environmental timeseries lifecycle. IoT restrained resources has an impact on data storage and dissemination, and IoT heterogeneity affects data acquisition and integration. Numbers correspond to book chapters.

the two IoT stressors against the processes of the environmental timeseries lifecycle. It also presents the structure of this thesis, which is comprised of six chapters. The core of this thesis is documented in Chapters two to five.

Chapter 2 inquires into **RQ1** by investigating the possibility of IoT prototyping devices to facilitate environmental timeseries lifecycle. We assessed the performance of an IoT prototyping device (i.e. Raspberry Pi), in terms of resilient data storage and processing power. To this end, we assembled a set of low-cost sensors, attached them on the Raspberry Pi and designed software to take observations under regular time intervals and disseminate them through standardized services, specifically OGC SOS. Then, we evaluated: a) resilient data storage by simulating power and network outages to assess persistence, b) processing power by conducting a stress test, which simulated concurrent users requesting for the stored observations. These tests provided us with insights regarding the capabilities of Raspberry Pi to operate in the weak IoT enabling ecosystem, and perform as an environmental data storage device.

Chapter 3 inquires into **RQ2** by investigating the ready-state of established timeseries dissemination protocols to operate efficiently in the IoT enabling ecosystem. We evaluated the operation of OGC SOS in a IoT setting, where Internet and/or power connection may be disrupted. As OGC SOS was not efficient and thus compatible to operate in the IoT ecosystem we designed and implemented a non-invasive extension, using on pagination. By-design our SOS pagination extension a) transforms OGC SOS to be disruption tolerant, b) supports for resource economizing, and c) maintains backwards compatibility. We validated the aforementioned by-design benefits by conducting experiments against different types of timeseries.

Chapters 4 and 5 inquire into **RQ3** by investigating declarative approaches to support e-scientists to acquire, transform and integrate syntactically (Chapter 4) and semantically (Chapter 5) heterogeneous timeseries datasets. Specifically, in Chapter 4 we designed a declarative method which allows e-scientists to describe a distinct dataset's syntax in an abstract manner through a template. We designed and implemented a template framework to cope with syntactic heterogeneity. We identified syntactic interoperability challenges, such as acquiring and integrating datasets with different formatting and diverse temporal and spatial references. We demonstrated the generality of our approach with several case studies spanning across different environmental domains (i.e. meteorology, agriculture, urban air quality and hydrology).

Chapter 5 investigates the semantic dimension of **RQ3**. We extended the declarative approach presented in Chapter 4 to perform semantic operations. E-scientists are enabled to annotate the semantics of the involved-in-a-dataset entities (e.g. observables, units of measurement, etc.) in a metadata file. We employed a reasoner which parses and stores the contents of the metadata files in a local ontology. This is used as a reference to determine compatibility among semantically heterogeneous datasets. We focused on one semantic heterogeneity challenge, that is the different units of measurement according to which observables are reported. We extended the implementation of the template framework which is reported in Chapter 4, in order to support the automatic transformation of the semantically heterogeneous, yet compatible datasets. We demonstrated our approach in a case study where we transform meteorological syntactically and semantically heterogeneous input files of four agricultural models, performing (when applicable) the on-the-fly units of measurement transformation.

Finally, Chapter 6 concludes this thesis, summarizes and discusses the findings from the previous chapters and proposes directions for future work.

Chapter 2

A Miniature Data Repository on a Raspberry Pi

This chapter is based on:

Samourkasidis, A., Athanasiadis, I.N. A Miniature Data Repository on a Raspberry Pi. Electronics 2017, 6, 1

Abstract

This work demonstrates a low-cost, miniature data repository proof-of-concept. Such a system needs to be resilient to power and network failures, and expose adequate processing power for persistent, long-term storage. Additional services are required for interoperable data sharing and visualization. We designed and implemented a software tool called Airchive to run on a Raspberry Pi, in order to assemble a data repository for archiving and openly sharing timeseries data. Airchive employs a relational database for storing data and implements two standards for sharing data (namely the Sensor Observation Service by the Open Geospatial Consortium and the Protocol for Metadata Harvesting by the Open Archives Initiative). The system is demonstrated in a realistic indoor air pollution data acquisition scenario in a four-month experiment evaluating its autonomy and robustness under power and network disruptions. A stress test was also conducted to evaluate its performance against concurrent client requests.

2.1 Introduction

Raspberry Pi has emerged as a key component in research, education and amateur cyber-physical systems. Raspberry Pi is a low-cost, mini-computer featuring processing, networking and video decoding capabilities (Upton & Halfacree, 2014). It has no permanent storage; the user may instead attach an SD card. It also exposes General Purpose Input–Output pins (GPIO) to connect with low-level peripheral devices through *Hardware Attached on Top* (HAT). Popular HATs include LEDs, motor controllers, sensors, and GPS devices (Nuttall, 2016).

Raspberry Pi has been developed primarily with the intention to encourage computer education in schools and the developing world, with the open philosophy in mind, as both the hardware design and operating system are open-licensed. Raspberry Pi has been demonstrated in a variety of applications beyond an educational context, including home-automation systems (Vujović & Maksimović, 2015), fire alarm systems (Bahrudin et al., 2013), home-security (Sapes & Solsona, 2016; Chowdhury et al., 2013), health supply chains monitoring (Schön et al., 2014), smart city applications (Jung et al., 2013; Leccese et al., 2014; Cagnetti et al., 2013) and environmental monitoring systems (Nikhade, 2015). Tanenbaum et al. (2013) viewed Raspberry Pi and similar technologies as enablers for democratizing technology and enabling creativity.

Despite the diversity of Raspberry Pi applications, little research has been done to investigate Raspberry Pi as a performing data repository. The low acquisition cost, the open hardware and software philosophy, and its capacity for interfacing with a variety of peripherals, renders Raspberry Pi a very good candidate for boosting open data, crowd-sourcing and citizen science movements. For instance, Raspberry Pi was employed to create a citizen observatory for water and flood management (Lanfranchi et al., 2014). Muller et al. (2015) discuss its potential use for crowdsourcing applications in climate and atmospheric sciences.

In this work, we present a proof-of-concept that Raspberry Pi can be used as a miniature, low-cost data repository that offers *persistent* data storage, and interoperable data sharing services over the Internet. We demonstrate *Airchive*, a system that stores and serves timeseries data recorded by a HAT equipped with air quality sensors, and investigate the system's robustness against power and network shortages. We also conducted a stress test in order to identify system limitations. The rest of the paper is structured as follows: in Section 2.2, we study the feasibility of the approach, by reviewing related work. Section 2.3 presents the overall system architecture, along with user types, system requirements, and key functionality. Section 2.4 presents the software platform developed and hardware utilized. Section 2.5 details our experiments with the system and presents the lessons learned, documenting difficulties and incidents arisen during the experiment period. Finally, Section 2.6 provides a discussion and lays the groundwork for future work. Section 2.7 provides a conclusion of the research.

2.2 Related Work

In principle, a data repository needs to offer persistent data storage, along with added-value services, as those for data processing, dissemination and visualization. Such services are similar to those offered by a Wireless Sensor Network (WSN) (Chang & Huang, 2016), an area where Raspberry Pi has been thoroughly investigated as a gateway node (or base station). A gateway node is the intermediate among sensor nodes and external networks. Its functionalities are regarded with (a) coordination (e.g. configuration of sensor nodes); (b) data storage; (c) data processing and (d) data dissemination to external clients (Dargie & Poellabauer, 2010). Most prominent advances in the usage of Raspberry Pi in WSNs have been done in the domains of (a), (c), and advanced data visualization.

Raspberry Pi has been used as a coordinator in a ZigBee mesh network interfacing with the World Wide Web. In (Ferdoush & Li, 2014), a Raspberry Pi performs as a gateway node and processes observations derived from the sensor nodes, stores them on a local database and provides visualization services to external users.

Data processing on the Raspberry Pi to offline calibrate sensor readings and provide data visualization is presented in (Lewis et al., 2016). Specifically, a Round Robin Database (Oetiker, 2014) was used for fast storage of sensor data with a constant disk footprint. This was done by keeping only the recent measurements in high resolution and statistical summaries for older recordings.

Advanced data visualization and image capturing is demonstrated in a volcanic monitoring system based on a Raspberry Pi (Moure et al., 2015). The Raspberry Pi creates and communicates graphs through commercial messenger applications—for example WhatsApp—while data are transferred daily to an external system for archival.

From the works above, it becomes clear that a Raspberry Pi may serve as a node that offers data storage, processing and visualization services, while still remaining a coordinating device interfacing sensors with the Internet. In most cases, data are forwarded to a remote, resourceful node in order to be archived in the long term. In this work, we aim to demonstrate that a Raspberry Pi can become an active archiver of its own sensor recordings, and investigate whether it is powerful enough to provide data storage and dissemination services on site.

2.3 The Airchive System

2.3.1 Objectives

Airchive (Samourkasidis & Athanasiadis, 2016) is a software product intended for being deployed on a Raspberry Pi to turn it into a self-contained data repository. *Airchive* provides data capture and dissemination services for timeseries measurements. There are two objectives in developing this system.

The first is to investigate long-term storage potential on a Raspberry Pi. The challenge here is inherited by the Raspberry Pi hardware limitations. *Airchive* provides with a persistent storage mechanism that is able to safe-keep its data in a trustworthy manner. We experimented this feature further, considering storage on both SD cards and USB disks attached with the Raspberry Pi.

The second is to demonstrate Raspberry Pi capacity to interoperate at the machine level through standard protocols for data sharing. *Airchive* adopts two mainstream standards to exhibit interoperability at the machine level. The first is the Sensor Observation Service (SOS), the Open Geospatial Consortium (OGC) standard tailored for sharing sensor observations (Bröring et al., 2012). SOS defines a Web service interface which allows querying observations and metadata of heterogeneous sensor systems. The second is the Protocol for Metadata Harvesting (OAI/PMH), an Open Archives Initiative low-barrier mechanism for repository interoperability (Lagoze & Van de Sompel, 2001). OAI/PMH is a generic protocol for sharing metadata among archives and has been widely adopted by digital libraries. Both SOS and OAI/PMH offer services that are invoked over the HTTP protocol.

2.3.2 Requirements

Airchive operates as a self-contained, autonomous repository for timeseries data archival and dissemination. It is a technical system that involves both software and hardware components, and needs to comply with certain non-functional requirements. From a software perspective, *Airchive* needs to be built with open-source tools and frameworks and be extensible, in order to respect the philosophy of the Raspberry Pi movement and maximize the potential for future uptake. Hardware support *Airchive* should be low-cost and resilient to power and network shortages. This will allow its use in remote locations, or in the developing world. The overall *Airchive* system should require low-technical skills to install, operate and maintain.

We identified three use cases for the *Airchive* system:

(a) **Web users** access the system through the Internet via a public webpage. They explore current or historical *Airchive* data, and they are interested in graphical representations of the content. Typically, a Web user is able to query for the data stored in *Airchive*, and the system will respond with a graph of the data requested. They may also download data in common formats, such as JSON (JavaScript Object Notation), CSV (Comma-separated values), GeoJSON (Butler et al., 2016) and GeoRSS (GeoRSS:).

(b) **Software agents** interact with *Airchive* for retrieving data or harvesting metadata. They may use different protocols and vocabularies to submit their requests. One may follow the SOS protocol for retrieving raw timeseries data, while another could use the OAI/PMH to get meta-information of the digital resources stored. Software agents interact with the system with RESTful Web services (Representational state transfer services) (Richardson & Ruby, 2008) over the HTTP protocol.

(c) The **system owner** has full access both locally and from the Internet via Secure Shell (SSH). Her responsibilities are to administer the system by updating system software or restarting the device.

Interoperability is an essential requirement of such a system. *Airchive* offers query services for software agents via SOS and OAI/PMH standards. SOS queries return responses in Extensible Markup Language (XML) using OGC vocabularies (as Observation & Measurements (O&M) (Cox, 2011), or Sensor Model Language (SensorML) (Botts & Robin, 2014)). OAI/PMH responses may be encoded in more than one metadata profile, including Dublin Core, a generic purpose metadata schema for annotating digital artifacts (DCMI Usage Board, 2012). By incorporating a variety of service offerings, we demonstrate the capabilities of a Raspberry Pi to operate with several clients, using different protocols and vocabularies, and support for **syntactic interoperability**.

Software development is based on our previous work reported in (Samourkasidis & Athanasiadis, 2014). We further improved the software system to host generic timeseries data. The current version has been thoroughly tested and is available as an open source software package (Samourkasidis & Athanasiadis, 2016). In this version, all of the metadata that are disseminated through OAI/PMH are calculated *on-the-fly* (instead of being stored permanently). This is a design choice to demonstrate the powerful processing power of Raspberry Pi.

Airchive can operate autonomously and with minimal user interventions. In the experiments discussed below, *Airchive* has been operating unattended for four months in order to evaluate its capabilities for *long-term* operation, as reliability, self-recovery and resilience to power and network failures.

2.3.3 Abstract Architectural Design

Airchive software platform was designed for the Raspberry Pi to turn it into a self-contained station for timeseries data archival and dissemination. It serves both real-time access and long-term storage and retrieval of sensor data, while also offering services for metadata harvesting. *Airchive* follows the *Sensing as a Service* paradigm (Perera et al., 2014) and is composed of five components that are implemented as loosely-coupled services, rendering the software highly extensible. The abstract architectural design is depicted in Figure 2.1.

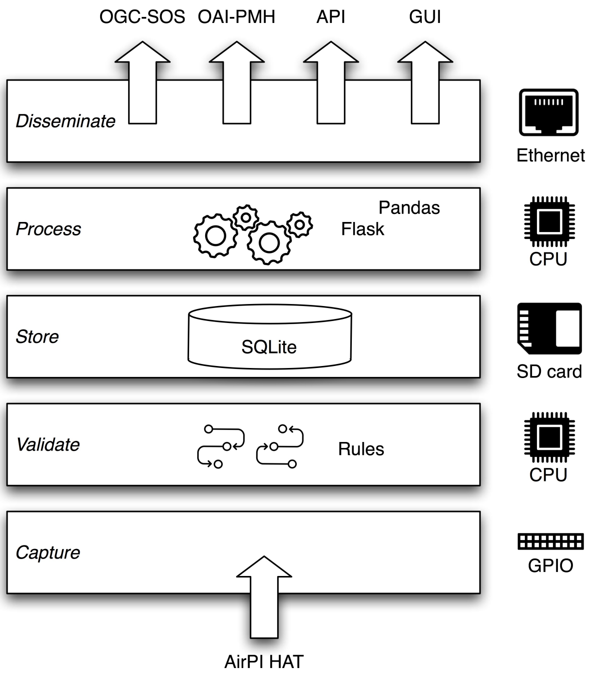


Figure 2.1: *Airchive* abstract architecture. System services are shown as layered components on the **left**, with relevant technologies. On the **right**, corresponding Raspberry Pi features are illustrated.

The data capture component (optional) comes first that actually collects sensed measurements from one or more sensor devices connected to the Raspberry Pi. This component is custom to hardware and/or sensors used. Our implementation interfaces with the sensors of the AirPi HAT. Nevertheless, the general behavior remains the same: at certain time intervals, it acquires the results from the sensors.

A data validation component (optional) may sit between data capture and data storage components. Its role is to apply quality assurance/quality control process and identify hardware or sensor errors. Additionally, it could associate the measurement with a quality flag by applying rules or more empirical procedures (i.e. statistical, data driven) (Athanasiadis & Mitkas, 2007; Athanasiadis et al., 2009, 2010; Athanasiadis & Mitkas, 2004). Such a component is essential for ensuring data reliability and user confidence.

The data storage component permanently stores sensor data in a relational database along with a time stamp. In order to be database-independent, an Object Relational Mapping (ORM) framework was utilized. The data storage component is also responsible for retrieving the data from permanent storage.

The data processing component is an intermediate layer between data storage and Web services. It transforms arguments (submitted by users/harvesters with their queries) into appropriate database queries, using the ORM framework. It also works in the other way around, as it formats database outputs according to user requests, using different formats (i.e. XML, JSON, CSV) or dictionaries (i.e. O&M, SensorML, Dublin Core). Finally, it offers descriptive statistics calculations *on-the-fly* (e.g. maximum, minimum, rolling mean, average and percentiles).

Last, but not least, the Web services components offer outlets for interaction with users and agents over the Internet. There are four Web service components in the current system, but more could be added in the future: *Web users* browse the repository and submit queries using a Graphical User Interface (GUI). *Software agents* interact with the SOS server, the OAI/PMH endpoint, or the *Airchive*'s own Application Programming Interface (API).

2.4 Implementation

2.4.1 Hardware

Airchive was deployed on a Raspberry Pi Model B. This model is equipped with a 700 MHz ARM processor, weights 45 g and has 512 MB of RAM. It is connected to the Internet through an Ethernet controller and features two USB ports. Instead of a hard disk, it uses an SD card. It is also equipped with 26 GPIO pins (General Purpose Input–Output) for interfacing with various peripherals (HATs). The chosen Operation System was Raspbian; a Linux based distribution for Raspberry Pi.

In order to generate data (i.e. actual observations) to be stored on the *Airchive*, we have chosen to use AirPi, a Raspberry Pi sensory HAT. AirPi is an interface board that connects over GPIO pins and is equipped with low cost air quality and weather sensors. It also follows the open hardware philosophy, and can be further extended with other sensors, including a GPS module (Dayan & Hartley, 2013). It costs roughly 90 USD including the sensors shown in Table 2.1. AirPi includes a software module that is able to log sensed data on the cloud.

Table 2.1: AirPi sensors with their respected observed properties.

Sensor Name	Observed Property	Type	Interface
DHT22	Relative humidity, Temperature	Digital	SPI
BMP085	Atmospheric pressure, Temperature	Digital	SPI
MICS-2710	Nitrogen dioxide	Analog	I2C
MICS-5525	Carbon monoxide	Analog	I2C

The overall system hardware is comprised of a Raspberry Pi Model B equipped with the AirPi HAT, an SD card and a USB memory drive, and it was connected to Internet with an ethernet cable, shown in Figure 2.2.

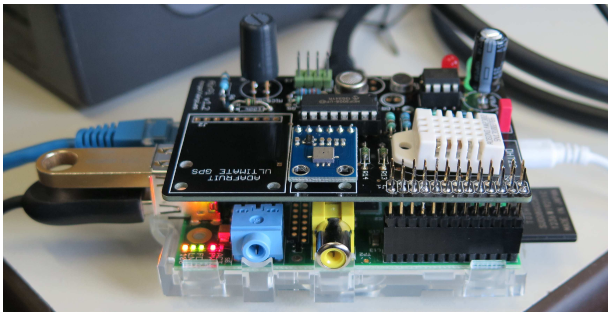


Figure 2.2: Raspberry Pi Model B with AirPi attached on top.

2.4.2 Software Development

Airchive software has been developed in Python, and is available as open-source software on GitHub (Samourkasidis & Athanasiadis, 2016) under the GNU Affero General Public License Version 3 (Free Software Foundation, 2016).

The data capture component interfaces with the AirPi libraries (Hartley, 2013) that transform electrical signals into human-understandable values. Data storage employs SQLite (Python Software Foundation, 2016), an open-source, lightweight relational database for Python and SQLAlchemy library (Bayer, 2016) for object-relational mapping. An outline of the data validation component is provided, but not fully implemented, as it is out of the core scope.

The data processing component was developed in three modules. The *Query* module handles client-requested queries and raises appropriate exceptions. Requests must include the sensor, the property and the corresponding timeframe for which the observations will be retrieved. A typical workflow is as depicted in Figure 2.3. The *Filter* module comprises of a set of statistical filters implemented as Python classes using pandas library (McKinney, 2011). Filters may be instantiated and applied *on-the-fly* on a query result. The *Format* module is responsible for serializing the query results. Jinja2 template engine (Ronacher, 2008) was used and the formats implemented correspond to the Web services offered. They include XML, GeoRSS, GeoJSON, JSON and CSV formats.

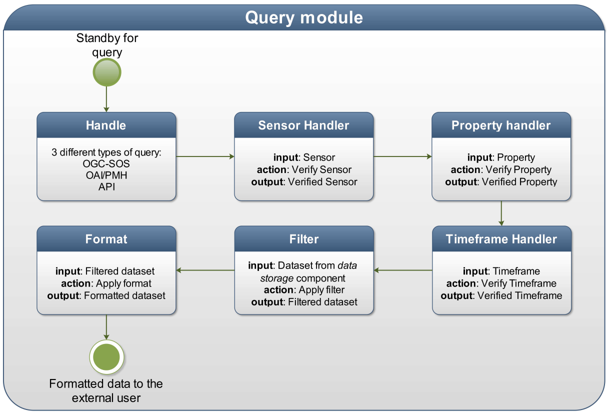

Query module
Standby for query
Handle
3 different types of query: OGC-SOS OAI/PMH API
Sensor Handler
input: Sensor action: Verify Sensor output: Verified Sensor
Property handler
input: Property action: Verify Property output: Verified Property
Timeframe Handler
input: Timeframe action: Verify Timeframe output: Verified Timeframe
Filter
input: Dataset from data storage component action: Apply filter output: Filtered dataset
Format
input: Filtered dataset action: Apply format output: Formatted dataset
Formatted data to the external user

Figure 2.3: A typical data processing workflow.

The Web service components were developed using the Flask web framework (Ronacher, 2010), in order to provide clients with static and dynamic content. Flask web framework deploys a web server which responds to `HTTP GET` requests with formatted data. Data requests can be submitted through the three API endpoints that we developed: the *Airchive* own API, SOS and OAI/PMH. A fourth outlet is the *Airchive* GUI, which is meant for the *Web users* to render graphs upon request. It uses the *Airchive* API for getting data, which are subsequently visualized on the client's web browser using Javascript. Graph rendering is facilitated by Flot (Laursen & Schnur, 2007) and jQuery (jQuery). Visualizing data occurs on the client browser, which also economizes resources of the Raspberry Pi.

Airchive software is generic in nature, in the sense that is does not require the data capture and validation components, and one could deploy it only with historical data. The system is configured via a file that aligns the timeseries with their semantics, including measured properties and units. The configuration allows for alternative definitions of the same observed properties, which enables the system to serve the same observations with a variety of vocabularies. Currently, we use the definitions of the Semantic Web for Earth and Environmental Terminology (SWEET) (Raskin & Pan, 2005).

2.5 Demonstration

2.5.1 Experimental Design

We deployed the *Airchive* system in a realistic scenario for indoor air quality monitoring. *Airchive* software was installed on a Raspberry Pi equipped with AirPi HAT, and installed indoors, connected to a power supply and the Internet via Ethernet port. We performed two experiments in order to evaluate the system autonomy and robustness, and its performance under load pressure.

2.5.2 Experiment 1: Autonomy and Robustness

In the first experiment, which lasted for more than 120 days, *Airchive* was exposed to irregular power and network disruptions. We did not interfere in system restoration during the "down" incidents and let the system self-recover.

In this experiment, a moderate sampling frequency to data capture component was set, in order to investigate the system's long-term storage capabilities. A measurement was retrieved from each sensor every 5 min. During this experiment, 183,850 measurements were gradually collected and served. In Table 2.2, there is a summary of the 24 network outage events observed during this period. Outage events were logged with UptimeRobot (UptimeRobot), a service that monitors web applications and notifies interested parties when an application is not accessible via the Internet. UptimeRobot was used only to log network failures, and did not interfere with our system.

Table 2.2: Statistics of the 24 the outage events, collected using UptimeRobot.com services.

Metric	Duration
Total downtime	10 days
Median downtime	7 min
Average downtime	543.6 min
St. dev. downtime	2572.4 min

During the first experiment, different users were submitting data queries to the system, in an ad-hoc manner, using the various interfaces: graph visualizations were requested by *Web users*, raw observations by SOS *clients* and derived metadata by OAI/PMH *harvesters*. We did not observe any malfunction for any of the client operations. Current and historical data were monitored, stored and disseminated appropriately, while the automated recovery worked as expected. OAI featured records were calculated *on-the-fly*, upon harvester requests in a timely fashion. We did not observe any notable delays in the capacity of the system to serve its clients.

2.5.3 Experiment 2: Stress Testing

During the second experiment, we conducted a stress test, in order to provide more insights regarding the system limitations. We investigated the number of concurrent user requests, after which the *Airchive* system delayed to respond. The sampling frequency was increased to 5 s. The experiment lasted for three days, and it collected and served more than 259,000 measurements. We utilized Locust (Heyman et al., 2011), an open source load testing tool written in Python. In Locust, a variable number of clients are deployed to submit concurrent requests to a service. Each Locust client submits a new request only when it receives a response to its previous request.

Locust takes as input the following parameters: (a) the number of concurrent clients; (b) the total number of requests; and (c) a url pointing to the requested resource. We set up three tests. In all cases, clients submitted a hundred requests altogether. The three tests involved the following requests over the Internet, via `HTTP GET`.

In the first test, clients request only the *Airchive* frontpage, which is a static HTML document. No transactions to the database were involved and the response size is constant (8740 bytes). Test 1 verifies that *Airchive* operates properly and examines if pressure on the Web services/dissemination components has an impact on the data capture component.

In the second test, clients request a set of 20 observations using *Airchive*'s API, and the response is formatted as a JSON document. This request requires an SQL query to be submitted to the database, and the response size is 538 bytes. Test 2 corresponds to the use case of a *Web user* that asks for a graph, as *Airchive* transmits the JSON document and the graph is rendered on the client-side.

In the third test, clients ask for the same set of observations as in Test 2, but this time over the SOS protocol, which returns an XML document. This request requires exactly the same SQL query to be submitted to the database but needs additional formatting for rendering the result in XML. The response size is 16,504 bytes. Test 3 corresponds to the use case of an SOS client asking data from *Airchive*.

We simulated four scenarios, in each of which we deployed a different number of concurrent clients. We tried one, five, 10 and 25 concurrent clients. This is a realistic assumption as the current system is not intended for large-scale deployment. We repeated the process five times for each test and scenario combination and reported two metrics in Table 2.3: (a) the average response time (ART) in *milliseconds*; and (b) the number of requests served per second (RPS).

Table 2.3: Experimental results for the three tests and for different numbers of concurrent clients. Average response time (ART) across a hundred requests for each document are reported in milliseconds. System throughput is expressed in requests per second (RPS).

Concurrent Clients	**Test 1: Static HTML**		**Test 2: API/JSON**		**Test 3: SOS/XML**	
	ART (std)	**RPS (std)**	**ART (std)**	**RPS (std)**	**ART (std)**	**RPS (std)**
1	66.4 (0.5)	16.2 (0.7)	1830 (27)	0.55 (0.01)	2171 (64)	0.46 (0.01)
5	344 (11)	15.4 (0.6)	9467 (178)	0.52 (0.01)	11,125 (90)	0.44 (0.00)
10	662 (7)	15.4 (0.3)	18,874 (397)	0.51 (0.01)	21,651 (281)	0.44 (0.01)
25	1576 (25)	16.6 (0.7)	45,607 (410)	0.49 (0.01)	49,427 (935)	0.45 (0.01)

Average response time (ART) is a proxy of the average delay to an external user request. For example, a user would have to wait 49.5 s (on average) plus the response time of their submitted request, under the scenario of the 25 concurrent clients for Test 3. As indicated by the results in Table 2.3, average response time is linearly correlated with the product of (a) number of concurrent users; and (b) average response time achieved when one client submits requests. We verify that requests per second (RPS) depend on the type of the requested document, and is rather stable regardless of the number of concurrent clients.

The introduced overhead to the system response times depends on the request and format type. Requests involving dynamic content are roughly 30 times slower than requests of static content. In the case of dynamic content requests, JSON-formatted responses are served 16% faster than the equivalent in XML.

Interpreting the results, we derive the number of concurrent (human) *Web users* that the system may serve. Assuming that a human user should not wait more than 6 s, we conclude that *Airchive* can serve simultaneously up to two *Web users* of the SOS/XML Web service (Test 3), or three *Web users* of the API/JSON Web service (Test 2). In the case of static content (Test 1), *Airchive* is able to serve up to 82 clients simultaneously. The numbers above do not represent *Airchive*'s maximum capabilities, rather its capacity for serving content to *Web users*.

In contrast, *software agents* interacting with such a system are usually not bound to any time limitation. We conducted further experiments to determine the threshold after which the system started failing to respond to requests. We increased the total number of requests to 500. We started increasing the number of concurrent users by multiples of 5, until requests started to fail. *Airchive* can serve simultaneously, without failure up to 254 (Test 1), 141 (Test 2) and 138 clients (Test 3). In excess of the client numbers above, the system continued to respond with more than one failure. The results are summarized in Table 2.4. These tests demonstrate *Airchive*'s capacity to work reliably with a significant number of clients.

We underline that despite the heavy workload we introduced during the stress tests, AirPi continued to operate normally. In all cases, we verified with the database content that observations were recorded every 5 s without any loss in all the experiments above (i.e. the data dissemination does not interfere with data capture).

Table 2.4: Estimates on the number of clients that *Airchive* can serve simultaneously.

User Types	Test 1: Static HTML	Test 2: API/JSON	Test 3: SOS/XML
Human Web users (response in less than 6 s)	82	3	2
Software agents (response guaranteed)	254	141	138

2.5.4 Incidents and Lessons Learned

During experiment 1, network failures occurred quite often. Those failures, impeded only Web connectivity and apart from the web server, the rest of the *Airchive* components continued to operate properly. We verified that no data loss occurred by cross-checking the time down intervals logged with UptimeRobot with the actual observations stored in the database.

We observed that the system was able to handle power failures, and it self-restored without human intervention. For all 24 outage events during experiment 1, *Airchive* recovered properly by making the Web service available as soon as the Internet connection returned. In this respect, the system demonstrated its persistence and credibility as a repository.

Calculating derived data (metadata) *on-the-fly* provided us with evidence regarding the system's extensibility and enhanced capabilities. Derived metadata, which were disseminated through OAI/PMH, were calculated upon client request. We observed that data were transmitted as fast as if they had been stored in the system. In addition, utilizing a Javascript framework for rendering graphs upon user request added no extra performance overhead to the Raspberry Pi. We did extensively evaluate these features with stress tests in experiment 2, and our experience was that the system performed as expected.

During experiment setup, we stumbled upon a recurring security incident. Given that Raspberry Pi was constantly connected to the Internet, it attracted malicious users after its first boot. We experienced a brute force attack to the SSH protocol that was trying to get unauthorized access to the device. We toughened up *Airchive* with a dedicated security software solution (fail2ban (Sumsal et al., 2005)), which prevented any further security incidents of that kind.

Another lesson learned had to do with a potential issue that may arise when power and networks fail at the same time. Raspberry Pi lacks a Real-Time built-in Clock (RTC), and it synchronizes its system clock through the Internet. In the case that an Internet connection is not available upon system boot, the Raspberry Pi system time is misconfigured. In the general use case of *Airchive*, this will not be a problem, but, in our experiments, this will result in errors in the data capture component, which will assign incorrect timestamps to data sensed from the HAT. This problem can be overcome so that the data capture component retrospectively reviews these timestamps when the Internet becomes available. An RTC HAT can be purchased and applied to Raspberry Pi. However, this option increases the total cost.

Last but not least, during the setup phase, we experimented with booting Raspbian and running *Airchive* from the USB disk instead of the SD card. First of all, this is a task that requires advanced technical skills and is still an experimental option not endorsed by the Raspberry Pi makers, and performance is not guaranteed. USB disks provide a cheaper storage option but are prone to failure. We experimented with this option for one month, during which the filesystem was corrupted twice, requiring the operating system and *Airchive* to be re-installed. Observed data were not permanently lost, but their retrieval required technical skills. In contrast, no such incidents occurred when the system operated on an SD card for a much longer period.

2.6 Discussion

Data persistence is a prerequisite for a data repository. In most efforts made with a Raspberry Pi and reported in the literature in the WSN context, data were periodically backed up in an external device and were not permanently stored on the embedded device. In the work presented here, *Airchive* relied solely on Raspberry Pi for permanent data storage. Our four-month experiments demonstrated that a Raspberry Pi equipped with an SD card can handle moderate and extensive read/write cycles without any issue; the resilience of SD cards is constantly evolving (Pinto, 2019).

The processing capabilities of Raspberry Pi have been investigated in the light of several applications. In *Airchive*, we studied its capacity to calculate and disseminate added-value data and indexes *on-the-fly*, i.e. upon user request. This way, less data are permanently stored and less write cycles are performed, which puts less pressure on SD card life.

Self-restoration from failures is another attribute of WSNs (Dargie & Poellabauer, 2010), which is also applicable in our work. Self-restoration contributes towards diminishing the technical skills that *Airchive* system owner should possess. During the experiments, the system self-restored from all power and network shortages that have been triggered, demonstrating that after its installation, the system can operate autonomously and without assistance.

We also consider that the *Airchive* system presented here also indirectly contributes to the *open data* movement, especially for the developing world. Besides the low acquisition cost and the low-technical skills required for its deployment, the system by-design responds to the "weak enabling environment" of the developing countries, i.e. intermittent, opportunistic Internet connection. In the frame of this work, we did not demonstrate the system in such conditions. However, we demonstrated that is able to attend to network and power failures.

Security and privacy are also two important attributes of a data repository system, and lay the foundation for future work. The *brute force attack* incident that occurred during the experiments is an illustration of the potential dangers. In addition, given that a data repository system may host personal and/or confidential data, more research should be focused on addressing privacy issues. There is a lack of any authentication mechanism, even in well-established, data dissemination protocols, such as OGC/SOS and OAI/PMH. An authentication mechanism can ensure privacy, and such issues should be addressed in the light of interoperable data dissemination on the application layer.

2.7 Conclusions

To summarize, we provided a proof-of-concept that current low-cost hardware is reliable enough to boost the *open data* movement. We demonstrated that a Raspberry Pi accompanied with an appropriate software can support persistent data storage, and provide added-value services on site. We designed and implemented an open-source, highly-extensible data repository software, called *Airchive*, to support data visualization, and interoperable data dissemination. We adopted two well-established data dissemination protocols: OGC Sensor Observation Service and Open Archive Initiative/Protocol Metadata Harvesting. Finally, we demonstrated its long-term data storage capabilities and resilience under harsh conditions of power and/or network failures, which take place irregularly. The load testing experiments provided us with insights about the Raspberry Pi performance under simultaneous requests from concurrent external clients.

Chapter 3

A Sensor Observation Service extension for Internet of Things

This chapter is based on:

Samourkasidis A., Athanasiadis I.N. A Sensor Observation Service Extension for Internet of Things. In: Podnar Žarko I., Broering A., Soursos S., Serrano M. (Eds.) Interoperability and Open-Source Solutions for the Internet of Things. InterOSS-IoT 2016. Lecture Notes in Computer Science, vol 10218. Springer, Cham

Abstract

This work contributes towards extending OGC Sensor Observation Service to become ready for Internet of Things, i.e. can be employed by devices with limited capabilities or opportunistic internet connection. We present an extension based on progressive data transmission, which by-design facilitates selective data harvesting and disruption-tolerant communication. The extension economizes resources, while respects the SOS specification requirement that the client should have no a-priori knowledge of the server capabilities. Empirical experiments in two case studies demonstrate that the extension adds little overhead and may lead to significant performance improvements in certain cases, as for irregular timeseries. Also, the proposed extension is not invasive and backwards compatible with legacy clients.

3.1 Introduction

Internet of the Things (IoT) is a dynamic, open, participatory ecosystem of decentralized and collaborative devices. Recent technological advances resulted in a plethora of low-cost devices with extended capabilities compared to traditional sensors. New generation of devices are miniaturized and empowered with storage, processing and networking capacity. They are essentially transformed into **smart nodes**, that operate autonomously, may offer added value services (Perera et al., 2014), and collaborate with each other in the cloud (Botta et al., 2016). Smart nodes could offer capture, storage and dissemination services of sensory information in a single device (Samourkasidis & Athanasiadis, 2017). IoT devices are also instrumental to the proliferation of new data sources (Hashem et al., 2015), sharing of information (Havlik et al., 2009), and contribute to the *big data* movement. Internet of Things advances the vision of Sensor Web, *an infrastructure which enables interoperable usage of sensor resources* (Bröring et al., 2011b). In the IoT era, Sensor Web is challenged to offer services that are interoperable, but at the same time perform efficiently with **less resources**, saving processing power and network bandwidth.

Interoperable data interchange for sensor data has been driven by the Open Geospatial Consortium (OGC). OGC introduced service interfaces and information models within Sensor Web Enablement (SWE), which is founded on machine-to-machine communication (Botts et al., 2008),(Bröring et al., 2011a). Service interfaces, as the Sensor Observation Service, Web Feature Service, Web Coverage Service, SensorThings provide interoperable means for geospatial information discovery and retrieval. Sensor Observation Service (SOS) (Na & Priest, 2007), (Bröring et al., 2012) is an OGC service interface, which promotes interoperable sensor-borne data exchange, operates as a web service, and supports for syntactic and semantic interoperability.

In the IoT era, architectural paradigms and technologies need to respect the limited capabilities of devices. The SWE 2.0 has been established with technologies as the Simple Object Access Protocol (SOAP) and XML-based information models, which are considered to add substantial overhead - a critical issue for IoT devices. On the other hand, Representational State Transfer (REST) and JSON-based information models seem to provide services which excel over SOAP and XML, in terms of power consumption and performance (Mulligan & Gračanin, 2009). Beyond these technical limitations, there are certain *design* choices that preclude SOS as an appropriate IoT outlet.

In this paper, we investigate current SOS design and propose an extension. In Section 3.2, we present related work, how SOS operates and challenges identified in the literature. In Section 3.3, we identify SOS design shortcomings from an IoT perspective, and introduce a pagination technique in order to promote selective data harvesting, enable seamless data integration and facilitate machine-to-machine interoperability. Section 3.4 presents an implementation and details the two case studies, which were designed to test the efficiency of the extension, along with experimental results. Section 3.5 provides with a discussion about our findings and contributions, concludes the research and lays the groundwork for future work.

3.2 Related work

3.2.1 Service orientation and interoperability in sensor networks

Service-Oriented Architecture (SOA) is an architectural paradigm founded on self-describing, self-contained services. Key concept in SOA is that services may be developed, maintained and served by different entities, and can subsequently be combined and produce composite applications. SOA has been instrumental for highly interoperable systems, as services are platform and language independent (Papazoglou & Georgakopoulos, 2003).

In the frame of interoperable data interchange, OGC introduced Sensor Web Enablement (SWE), which follows the SOA architectural paradigm. Standards developed within SWE provide means for the discovery and retrieval of sensor observations. SWE contributes towards the vision of Sensor Web, where web-accessible sensor networks and archived sensor observations can be discovered and accessed using standard protocols and application program interfaces (APIs) (Botts et al., 2008). They are realized through *web services*, i.e. services "identified by a URI, whose service description and transport utilize open Internet standards" (Papazoglou & Georgakopoulos, 2003). Communication between service interfaces and other services or clients is achieved through Simple Object Access Protocol (SOAP), which builds on existing communication layers (i.e. HTTP) (Curbera et al., 2002). SWE is a very important infrastructure (Bröring et al., 2011b) as it offers interoperable protocols for advertising, disseminating and requesting data among heterogeneous sensor systems and devices.

3.2.2 The Sensor Observation Service

Sensor Observation Service (SOS) is an OGC service interface specification for accessing sensor observations, which acts as "the intermediary between a client and an observation repository" (Botts et al., 2008). SOS interface enables clients to request, filter and retrieve observations, and metadata about repositories and sensors.

SOS comes with a *core* set of services, and *extensions* that enrich it with extra functionality, or *profiles* for domain-specific behavior. The current 2.0 specification (Bröring et al., 2012) defines three *core* operations:

a. service discovery (`GetCapabilities`),
b. sensors metadata retrieval (`DescribeSensor`), and
c. observations retrieval (`GetObservation`).

There are several extensions and profiles available, but their description falls outside the scope of this paper. As an indicative example for the reader, the *transactional extension* provides with services to register new sensors and add new observations.

SOS is a pull-based service interface and is intended for machine-to-machine communication. The protocol prescribes a communication between a client and a server, both can be considered to be software agents. The client submits a request and the server answers with a response, typically in the form of XML document. Responses are encoded in appropriate SWE related

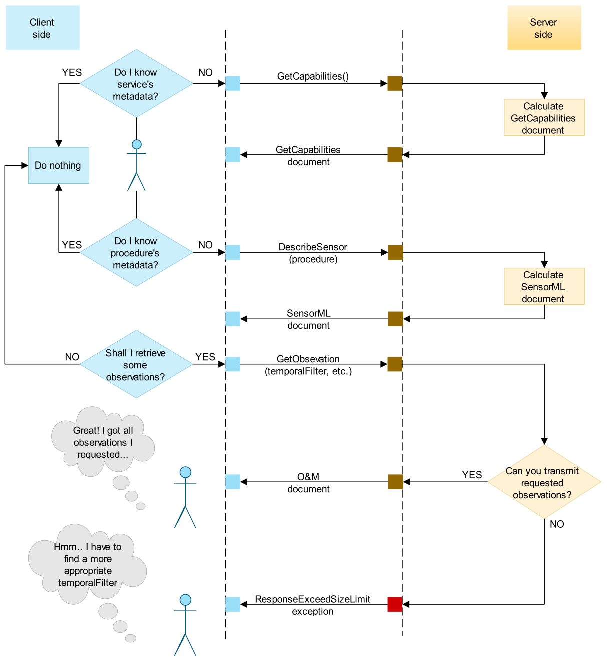

Client side
Server side
Do I know service's metadata?
YES
NO
GetCapabilities()
Calculate GetCapabilities document
GetCapabilities document
Do nothing
Do I know procedure's metadata?
YES
NO
DescribeSensor (procedure)
Calculate SensorML document
SensorML document
Shall I retrieve some observations?
NO
YES
GetObsevation (temporalFilter, etc.)
Great! I got all observations I requested...
O&M document
YES
Can you transmit requested observations?
NO
Hmm.. I have to find a more appropriate temporalFilter
ResponseExceedSizeLimit exception

Figure 3.1: A typical observation retrieval workflow using SOS

XML schemas as Observation & Measurements (Cox, 2011), or SensorML (Botts & Robin, 2014). A typical observation retrieval workflow using SOS is depicted in Figure 3.1. First, the client inquires the server for its capabilities. Then, it may ask for descriptions on certain sensors, and finally requests for observations from one or more sensors. A typical `GetObservation` request includes temporal and/or spatial boundaries.

When SOS server encounters an error while performing a `GetObservation` operation, it returns an exception. For example, if client asks for wrong values of arguments an `InvalidParameterValue` exception is rendered. In the current SOS 2.0 interface standard (Bröring et al., 2012) there is also a type of exception for the cases that the response exceeds a **size limit**. We will investigate this further below.

3.2.3 Challenges in sharing sensor observations in IoT

Internet of Things consists of *smart nodes* equipped with sensors and network connectivity, able to interact with their environment and share information. *Smart nodes* are entitled with specific characteristics:

a. restrained capabilities (in terms of energy and processing power),
b. opportunistic Internet connection, and
c. heterogeneity in resulting data formats and communication protocols (Atzori et al., 2010).

Key challenges towards the IoT realization include energy efficiency, integration of service technologies and security/privacy (Li et al., 2015). Also, thematic and spatial concerns of deployed IoT systems pose great challenges in spatiotemporal aggregation of disperse observation datasets.

As regards with **heterogeneous sensor integration**, previous studies have been conducted towards various directions. A virtual integration framework for heterogeneous meteorological and oceanographic data sources is demonstrated in (Regueiro et al., 2015). A *SOS profile* to facilitate multi-agency sensor data integration was reported in (Jirka et al., 2012; Alamdar et al., 2016; Fredericks et al., 2009) argue in that quality metadata should also be transmitted through SWE services, in order the realization of automatic data integration to be achieved.

Integration of spatially diverse sensor timeseries utilizing OGC standards concerned Horita et al. (2015). They developed a spatial decision support system for flood risk management, associating Volunteered Geographic Information (VGI) and measured data derived from Wireless Sensor Networks (WSNs). Data acquisition, integration and dissemination is orchestrated by a SOS instance.

Only recently, OGC introduced *SensorThings API* to facilitate "the interconnection of IoT devices, data, and applications over the Web" (Liang et al., 2016). In contrast with other OGC standards, SensorThings API adopts the REST paradigm and utilizes JSON-based information models. SensorThings API defines HTTP requests to facilitate observations' retrieval, as well tasking of sensors and actuators.

Using parameters to regulate response size to requests within OGC-related standards, was a topic of interest for (Volker Andres, 2014), (Vretanos, 2014) and (Liang et al., 2016). Lengthy responses to `GetObservation` requests have been identified as a potential danger to both SOS server and clients (Volker Andres, 2014). In the same work, it has been indicated that beyond the `ResponseExceedSizeLimit` exception, other certain limitations as regards with the number of returned observation should be concerned and imposed. The WFS interface standard (Vretanos, 2014) and the SensorThings API offer a paging implementation, that allows the client to limit the number of features included in a response by using two optional arguments (*count*, *startindex* for WFS, and *top*, *skip* for SensorThings API).

Last but not least, several researchers investigated the suitability of limited bandwidth, energy, and processing power devices to host a SOS server. These have mainly concentrated on (a) adoption of lightweight architectural paradigms (e.g. REST instead of SOAP (Rouached et al., 2012), (Janowicz et al., 2013), (Yazar & Dunkels, 2009)), and (b) evaluation of SOS lightweight implementations (Pradilla et al., 2015), (Jazayeri et al., 2015). We have also deployed SOS over a Raspberry Pi to exploit the potential of low-cost embedded devices (Samourkasidis & Athanasiadis, 2017).

In this work we concentrate on the SOS service interface design and evaluate the efficiency of communication between client and server.

3.3 Methods

3.3.1 SOS service interface design issues

According to SOS specification, clients are not allowed to know sensor observations' frequency. The server advertises the boundaries of the information it holds, but not the resolution. Any client is not possible to infer the sensor temporal or spatial resolution, based on their communication with the server. This requirement is that the client has access with **no a-priori knowledge** (Na & Priest, 2007). While this enforces reusability and generality of the service interface, it may may lead to excessive data requests, which may result to server overload, or even Denial of Service attacks.

Excessive data transmission has been identified as an issue for `GetObservation` requests. In the first specification of SOS, there was not imposed any limitation, regarding the maximum number of observations which could be transmitted. For the server, the only viable response to of a `GetObservation` request was to return a set of observations. The server had no way to refuse to respond, in cases where the client was asking for an excessive amount of data, it was busy, or any other reason.

To illustrate the above shortfall we will consider a service offered by National Oceanic and Atmospheric Administration (NOAA) (Center for Operational Oceanographic Products and Services (CO-OPS), 2019). NOAA's Center for Operational Oceanographic Products and Services (CO-OPS) offers openly a variety of sensor observations using SOS. In this implementation, if a client requests observations for a time range which exceeds 31 days, the server responds with an exception, rejecting the parameter value:

```
<Exception exceptionCode="InvalidParameterValue"
    locator="eventTime">
        <ExceptionText>
            Max 31 days of data can be requested.
            62.0 days were requested.
        </ExceptionText>
</Exception>
```

Note that the exception rejects the parameter value, disclosing in a non machine interoperable message of the size limits for this request.

In the future work section of SOS 1.0 specification (Na & Priest, 2007) it was acknowledged that: *"The density of requests and offerings must be addressed,... so that large data volumes are not transmitted unnecessarily due to a lack of information about service offerings."*. Indeed, that was addressed in SOS 2.0 by introducing an *exception* to manage excessive data requests, while taking into consideration the *no a-priori knowledge* requirement (Botts et al., 2008). The `ResponseExceedSizeLimit` exception functionality resembles the response of NOAA server above, but with pertinent semantics to the exception thrown: The server is able to inform the client that the *"requested result set exceeds the response size limit of the service and thus cannot be delivered"* (Bröring et al., 2012). Both server and client applications are protected from extremely big response sizes, and the *no a-priori knowledge* requirement is respected.

The `ResponseExceedSizeLimit` exception of SOS 2.0 is a significant improvement compared to SOS 1.0, as it allows the server to respond to a request with an exception than with actual data. Note that, the response size limit should not be considered a fixed parameter. It could change when there is high traffic, or service maintenance. In those conditions, the server should be allowed to not to respond to requests that would under normal conditions.

However, the main limiting factor to this design is that clients have no insights regarding the carrying capacity of the server, or (equivalently) the density of an offering. Due to the *no a-priori knowledge* requirement, clients cannot infer how to narrow down their requests so that server responds.

We identify two cases here. First case is when the server publishes regular sensor observations. Under this category fall most long-term, permanent sensor infrastructures. In this case, clients could implement heuristic techniques to discover the response size limit (assuming that it is constant).

In the second case, observation streams are irregular. This may happen if the sensor sampling frequency varies, or sensors move. For example, consider sensors operating in energy restrained environments and adopt opportunistic sensing techniques, or event-based sensing (de Assis et al., 2016). Volunteered Geographic Information Systems which enable individuals (Goodchild, 2007),(Drosatos et al., 2014) or cars (Bröering et al., 2015) as data providers, fall in the same case. In these situations, it is impossible for the client to make any kind of estimate on the response size, and devise a strategy to reduce accordingly the spatiotemporal boundaries of their query.

Responding with an exception to voluminous requests could be tolerated in fixed sensor networks (case one above). However, it hinders SOS applicability in resource-constrained environments. As clients are neither aware of the response size limits, nor how to restrict their queries, the SOS communication protocol underperforms: It wastes both processing power and network bandwidth as it is engaged in more request/response cycles. This, ultimately results in bigger response times. Such drawbacks are incompatible with the Internet of Things needs. This problem could be addressed by introducing a progressive data transmission technique described below.

3.3.2 The resumption token technique and Open Archives Initiative

The notion of selective data retrieval was introduced in Lagoze & Van de Sompel (2001). Utilizing a *resumption token*, large and resource-demanding data transactions are fragmented into several requests/responses. The client submits a request and the server responds with a part of the result and a *resumption token*. Then the client (harvester) can use this *resumption token* in follow up requests to get the following part of its initial request. Gradually, by consecutive requests the client retrieves the all the partial answers to its initial request. This mechanism enables the server to handle with requests that have large responses, with respect to available bandwidth and/or processing power.

3.3.3 A pagination extension for SOS

SOS service interface can address IoT needs by introducing progressive data transmission. We extend the current SOS service interface with a *resumption token* parameter in the `GetObservation` requests. By fragmenting requests into many sequential ones, we transform SOS into a **disruption-tolerant** service interface, as clients are enabled to ask for specific observation subsets. Observations are divided and loosely packed into *pages* of certain size. The number of observations contained in a *page* (i.e. chunk of subsequent observations) is determined by the SOS server.

The observation retrieval workflow according to the proposed design is depicted in Figure 3.2. The client asks for a set of observations with a `GetObservation` request. The server processes the request, and always responds with an O&M document. If the response exceeds the carrying capacity of the server, results will be organized in subsets (called pages), and the server response will include an additional element, called `next` which will point to the URL of the next page of results. The next page URL is the same as the original request, but contains an extra parameter called page, which has the role of the resumption token. The `page` parameter is optional: when a client request does not contain a `page` argument, the server responds with the first *page* of the request. The last page of the parts contains no next page element to notify the client of the end of the transmission.

In the simplest case, server carrying capacity could be an arbitrary, fixed threshold, similar to the *request size limit* of the SOS 2.0 exception. Of course, the server carrying capacity may dynamically vary according to result set properties, or server resources, enabling network load balancing, efficient use of energy, etc. It could even change during the transmission, as the total number of pages is not disclosed to the client. The `page` resumption token could be constructed incrementally as page number in case the server has a fixed carrying capacity, and data do not change. In case of varying page size, the `page` parameter can take unique pseudo-random integer values. In case where data changed during the communication, or any other reason, the next `page` token could be revoked by the service provider.

3.3.4 Expected (by design) benefits

The paginated protocol proposed here is beneficial for both server and client **efficiency** and **performance**. The communication protocol does not waste resources to respond with exceptions, as all requests result to responses that carry observations. This saves processing power and communication bandwidth in both client and the server.

Another attribute of the design we propose is its **non-invasive** nature. Given the *page* parameter is optional, current SOS clients can seamlessly submit `GetObservation` requests and retrieve observations, as long as the SOS server carrying capacity is not exceeded. This means that existing SOS 1.0 or 2.0 server infrastructures could switch to a paginated implementation, and as long as they do not change their size limit threshold, existing clients would continue to operate without disruption. In the rest cases, a *page-parser* method should be implemented and incorporated in legacy clients. This method would parse a `GetObservation` response document to determine the URL of the next `GetObservation`

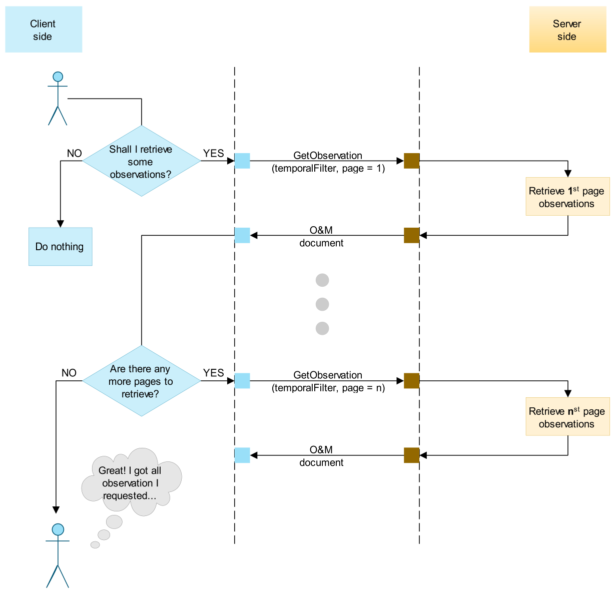

Client side
Server side
Shall I retrieve some observations?
NO
YES
Do nothing
GetObservation (temporalFilter, page = 1)
Retrieve 1st page observations
O&M document
Are there any more pages to retrieve?
NO
YES
GetObservation (temporalFilter, page = n)
Retrieve nst page observations
O&M document
Great! I got all observation I requested...

Figure 3.2: A typical *paginated* observation retrieval workflow

request. On server side, the *pagination* extension could be easily applied on top of existing implementations.

3.4 Demonstration and implementation

3.4.1 Setup

The SOS pagination extension introduced above comes with design advantages discussed in the previous section. There are also performance improvements that we experimentally evaluated by setting up two case studies. Without loss of generality, we assume not movable sensors that hold timeseries information. In case study one, the server holds a regular timeseries dataset, while in the second case study an irregular one. For both cases, we compared the SOS pagination extension (**SOS-p**) service interface against *SOS 2.0*.

The *SOS-p* server is queried by a corresponding client (*PAC*), that is able to handle `page` resumption tokens. For *SOS 2.0* server, we considered two clients: one that is not aware of *SOS 2.0* carrying capacity and finds it by employing a divide-and-conquer algorithm (*DAC*); and one that has this a-priori knowledge (*LEC*).

The three clients are in detail as follows:

Divide and Conquer client (DAC): *DAC* submits `GetObservation` requests according to *SOS 2.0* specification. When the server responds with a `ResponseExceedSizeLimit` exception, *DAC* halves the time window and submits a new query. When *DAC* finds a time window for which the server responds with no exception, it continues asking for observations with of this duration size in the temporal filter, until it has received all the data corresponding to the original request.

Leaky client (LEC): *LEC* knows the server *carrying capacity* and arranges the *temporal filter* of its request, so that there are no exceptions. While this is against the *no a-priori knowledge* requirement, it corresponds to the most favorable situation for the existing *SOS 2.0* protocol. *LEC* submits `GetObservation` requests to *SOS 2.0* only for case study 1.

Pagination-aware client (PAC): *PAC* client submits `GetObservation` requests according to *SOS-p*, i.e. it is capable of processing the page resumption token. In its first `GetObservation` request asks for the first page, and then processes the response for the next `page` it will ask for. If the `GetObservation` response document does not contain a next *page* tag, it means that all requested observations were transmitted.

3.4.2 Implementation and synthetic datasets

This study makes use of the AiRCHIVE SOS server implemented in Python (Samourkasidis & Athanasiadis, 2017). Clients were also implemented in Python. Queries to SOS server were submitted as *HTTP GET* requests via Python Requests module (Reitz, 2017). Response times for each case study were facilitated using the Python Time module (Python Software Foundation, 2017b). All experiments were carried out on a Intel Core i5 4 Mac with a 2,4 GHz and 16.0 GB of memory (1600 MHz DDR3), running OS X El Capitan (Version 10.11.1). SOS server and SOS client instances operated on the same physical machine.

In both case studies, a dataset of 15'000 observations was artificially generated. In case study one, we assumed that measurements are sensed in constant intervals of 10 seconds. In case study two, observations were timed with a *inconstant* frequency. Observation time interval varies from 10 to 3000 seconds, distributed uniformly. Timestamps were generated with the Python Random Number generator module, using Mersenne Twister (Python Software Foundation, 2017a). Both timeseries were stored in two SQLite databases and made available to the servers.

3.4.3 Experimental setup and metrics

As limited bandwidth and processing power are key elements of IoT systems, we set up accordingly our experiments. The **carrying capacity** of the servers was defined to be 15 observations. This arbitrary threshold was chosen so that there will be significant traffic of SOS requests. *SOS 2.0* server would render a `ResponseExceedSizeLimit` exception if the result set would include more than 15 observations. *SOS-p* server organizes its responses in pages of 15 observations per page.

Clients were configured to request for observations for time intervals that result to 1 000, 2 000, 4 000, 8 000 or 15 000 observations (*response length*). Experiments have been repeated 10 times for all clients and both case studies.

For all experiments, we recorded two metrics:

a. the **response time** is the total time passed until the client has received the total amount of data requested. Measured in *seconds*.
b. **transfer volume** is the total size of all response documents received by the client until the whole response has been received. It is measured in *MB*.

Response times are averaged across the 10 repetitions, while transfer volume is the same for each repetition.

For the cases of **SOS 2.0** implementation, in the *average response time* and *transfer volume*, time spent and resulted size of exceptions are also included.

3.4.4 Experimental results

Tables 3.1 and 3.2 summarize the results for both case studies and all clients. The *response time* is reported as average and standard deviation of ten repetitions.

For case study 1, best results are achieved, as expected, by the client that is aware of the server carrying capacity (*LEC*), but violates the *no a-priori knowledge* requirement. The divide-and-conquer client (*DAC*) in *SOS 2.0* adds an overhead to the transmission, as it needs to search for a working time interval. Its performance is affected mostly of how close the time interval found is to the servers carrying capacity. The response time was significantly increased in our experiments in Table 3.1. In the contrary, the performance of *SOS-p* and the paginated client *PAC* is very close to the server carrying capacity, without any breach of the *no a-priori knowledge* requirement. Experimental results in Table 3.1 illustrate overheads less than 5% in response time for up to few hundreds of pages, while for bigger numbers of requests overheads in time may end up to 30% in response time. This is attributed to the efficiency of the pagination implementation and is a well-known limitation among the database community. In the future, we will investigate other database options that can improve this further.

Table 3.1: Experimental results for the **regular** timeseries for all three clients. Average response times and standard deviation across ten requests are reported. Total volume of the data transmitted, number of requests, and number of exceptions for *DAC*.

Query	**PAC**			**LEC**		**DAC**		
Length	resp.time(std) [s]	vol [MB]	reqs	resp.time(std) [s]	vol [MB]	resp.time(std) [s]	vol [MB]	exceptions
1000	1.34 (±0.049)	0.59	67	1.29 (±0.013)	0.58	2.36 (±0.02)	0.63	7
2000	2.68 (±0.012)	1.2	134	2.57 (±0.011)	1.2	4.64 (±0.05)	1.3	8
4000	5.53 (±0.083)	2.4	267	5.22 (±0.017)	2.3	9.27 (±0.03)	2.5	9
8000	11.93 (±0.031)	4.7	534	10.31 (±0.034)	4.7	18.31 (±0.06)	5.0	10
15000	24.77 (±0.739)	8.9	1000	19.05 (±0.043)	8.7	21.33 (±0.06)	8.8	10

For case study 2, irregular timeseries are served therefor there is no notion of leaking the prior knowledge of the server carrying capacity. Here the paginated *SOS-p* excels over *SOS 2.0*, as presented in Table 3.2. *SOS-p* and *PAC* are faster than *SOS 2.0* by more than 60% on average on every `GetObservation` request. Also, note that number of requests has been roughly doubled, which results to a noticeable difference in the amount data transmitted.

Table 3.2: Experimental results for the **irregular** timeseries. For *PAC* and *DAC* clients reports average response times and standard deviation across ten requests. Total volume of the data transmitted, number of requests and number of exceptions for *DAC*.

Query	**PAC**			**DAC**		
Length	resp.time(std) [s]	vol [MB]	reqs	resp.time(std) [s]	vol [MB]	exceptions
1000	1.35 (±0.02)	0.59	67	2.40 (±0.03)	0.63	7
2000	2.75 (±0.05)	1.2	134	4.71 (±0.05)	1.3	8
4000	5.66 (±0.07)	2.4	267	9.28 (±0.06)	2.5	9
8000	11.97 (±0.08)	4.7	534	18.34 (±0.03)	5.0	10
15000	24.62 (±0.11)	8.9	1000	36.81 (±0.88)	9.5	11

This is to be expected, as the *divide and conquer* strategy may end up finding a query window that is far from what can be actually served. There could be other search algorithms employed for improving *DAC* performance. However, it is made clear from this experiment, that the paginated protocol guarantees **by design** that the optimal number of measurements is included in each response. *SOS-p* entrusts the burden of coordinating the observation boundaries to the server, which knows its limits, than having the client wasting resources with requests of suboptimal lengths. The improved performance ensures that there is no waste of resources on both the client and the server side.

3.5 Discussion and Conclusions

This work contributes towards improving OGC SOS protocol to become IoT ready. Drafting on top of IoT requirements as efficient resource utilization and opportunistic Internet connection, and taking into consideration response size to `GetObservation` requests requirements set in Volker Andres (2014), we designed a SOS extension, which implements a *pagination* mechanism.

There is a fundamental difference between our design and the paging mechanism introduced in OGC WFS (Vretanos, 2014). WFS paging design contradicts with the rationale of SOS `ResponseExceedSizeLimit` exception, that is to enable the SOS server to manage efficiently its resources. Conversely, it allows clients to select the number of returned observations, which is a feature that can only facilitate specific applications (e.g. Graphical User Interfaces which can visualize a certain number of observations). In the contrary, the solution proposed in this work follows the Open Archives Initiative design pattern, and the decision on the *page size* remains with the server, not the client. As we demonstrated above, this is a necessary condition for the server in the IoT era, as it allows for parsimonious use of resources, and protection from queries resulting with very big results.

Pagination introduces the notion of **progressive transmission**, which fits for purpose with timeseries data sequential nature, but is also suitable for any kind of spatiotemporal requests. It adds **disruption-tolerance** as an additional SOS feature, since a client can request for and retrieve a specific page. This is very useful when big datasets are to be transmitted or when the Internet connection is poor. Our design enables a SOS server to exploit its resources to the maximum, as computational power and network bandwidth are spent for yielding results, not for handling exceptions. Thus, the paginated extension enables **by-design** SOS for devices with restrained capabilities, where resources are economized in sharing interoperable knowledge.

Whilst our suggested design entails new improvements to the existing *SOS 2.0*, its importance is highlighted by its **non-invasive** nature. Backwards compatible design is achieved through the *optional* page parameter, since all requested data could be included in one *page*. This way, current *SOS 2.0* clients could operate without further modifications with *SOS-p* extended servers, if the server always responds with the whole data requested.

Evaluating the *SOS-p* extension against specific metrics, we validated improvements by experiments. Those improvements are mainly concerned with efficiency. Lower `GetObservation` requests completion times contribute towards IoT devices energy conservation, since computational resources are occupied for less time, and thus more clients can be served simultaneously. In addition to that, when carrying capacity is not known to the client, the *SOS 2.0* protocol is under-operating, as possibly transmits less observations in each request. This results to more request-response transactions, with overheads in data volume and duration time.

The pagination extension introduced here offers a remedy to *SOS 2.0* shortfalls in handling exceptions, by providing a machine interoperable solution. It also fills-in the SOS missing piece, that is to "*allow a client to determine the density of an offering*" (Na & Priest, 2007). Instead of that, it delegates to the server to drive protocol.

Advancements discussed so far lay the groundwork for future work. Firstly, our intention to use *pagination* was exploratory, thus there is room for further improvements in the implementation to further improve performance. One direction for improvement is the adoption of a caching mechanism. *Pagination* is a good candidate for caching techniques, since requests are incremental and queries are submitted sequentially. With the design introduced here, the client reveals its intentions to the server, by asking the whole spatiotemporal boundaries of interest. If the response is too big, the server will return the first page that includes a part of the results. As the client intentions have been disclosed to the server, this allows for caching mechanisms to be set up on the server side.

To summarize, we argued that current SOS design was not intended for the Internet of the Things era. We designed a pagination extension offering progressive data transmission, economizing resources and tackling with limited or interrupted Internet connectivity with a disruption-tolerant protocol, while respecting SOS specification. There is a small effort into extending current SOS servers and clients to implement the pagination extension, while there are significant performance improvements, as indicated by the experimental results. The pagination extension sets the grounds for enabling SOS as an Internet of the Things dissemination outlet for sensor observations.

Chapter 4

A template framework for environmental timeseries data acquisition

This chapter is based on:

Samourkasidis, A., Papoutsoglou, E., Athanasiadis, I. N. A template framework for environmental timeseries data acquisition. Environmental Modelling & Software, 117, 237–249.

Abstract

This work demonstrates a template framework for acquiring and integrating heterogeneous environmental timeseries. Internet of Things contributes towards the high-availability of environmental timeseries datasets. These are available through different protocols and stored under diverse, custom formats, rendering data acquisition and integration a laborious process of the environmental data lifecycle. We designed and implemented a template framework, called EDAM to facilitate diverse data acquisition and integration. EDAM is founded on re-usable templates, and requires no strong computer science background. EDAM supports for data dissemination in custom formats, as instructed by output templates. A template is an abstract representation of a data file's structure written using programming language agnostic semantics. We demonstrate EDAM generality, by scrapping online meteorological data, extracting observations from a relational database, and aggregating historical timeseries stored on local files.

4.1 Introduction

Environmental data management, that is acquisition, processing, storage, and dissemination (Athanasiadis & 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 & Robin, 2014), WaterML 2.0 (Taylor, 2014) etc.). The various datasets need to be stored in a common schema 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 & 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 & Trefethen, 2003) which "promotes innovation in collaborative, computationally- or data-intensive research across all disciplines, throughout the research lifecycle" (IEEE International Conference on eScience, 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 & Drake, 2003), R (Ihaka & Gentleman, 1996), etc.) and data models (e.g. O&M, WaterML 2.0, etc.) are adopted for the *scripting* and *environmental data management frameworks* 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 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 (Bröring 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:

a. scraping meteorological data from the public webpages of the Bureau of Meteorology (BoM) in Australia and the UK Met Office,
b. parsing hydrological timeseries data from the Hydrological Observatory of Athens (HOA),
c. extracting observations from an air quality archive from BoM, originally stored in a relational database,
d. aggregating historical timeseries data from all Dutch weather stations, provided by Koninklijk Nederlands Meteorologisch Instituut (KNMI),
e. 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 4.2 we review contemporary approaches for environmental data acquisition and integration, and introduce readers to environmental data management with web template frameworks. Section 4.3 presents the *EDAM* architecture; specifically: key requirements, user types, and use scenarios. Section 4.4 demonstrates *EDAM*, the conducted experiments, the used datasets and the addressed challenges. Finally, in Section 4.5 we discuss our research findings and lessons learned, conclude the research summarizing key findings and future work.

4.2 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 4.4.1).

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 4.2.2 we present the scripting approach which facilitates environmental data transformation to fit into a consistent data format. Last, Subsection 4.2.3 introduces web template frameworks and presents our previous experiences with them.

4.2.1 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) (Bröring 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 (Bröring 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 & 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.

4.2.2 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. (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.

4.2.3 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, 2007), Cheetah3 (Broytman & 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 & 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.

4.2.4 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 & 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.

4.3 The EDAM framework

4.3.1 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 4.2.2). 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.

4.3.2 Abstract architectural design

There are three key-components involved in *EDAM*: a) input files, b) template files, and c) template engine. Figure 4.1 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.

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. Figure 4.2 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

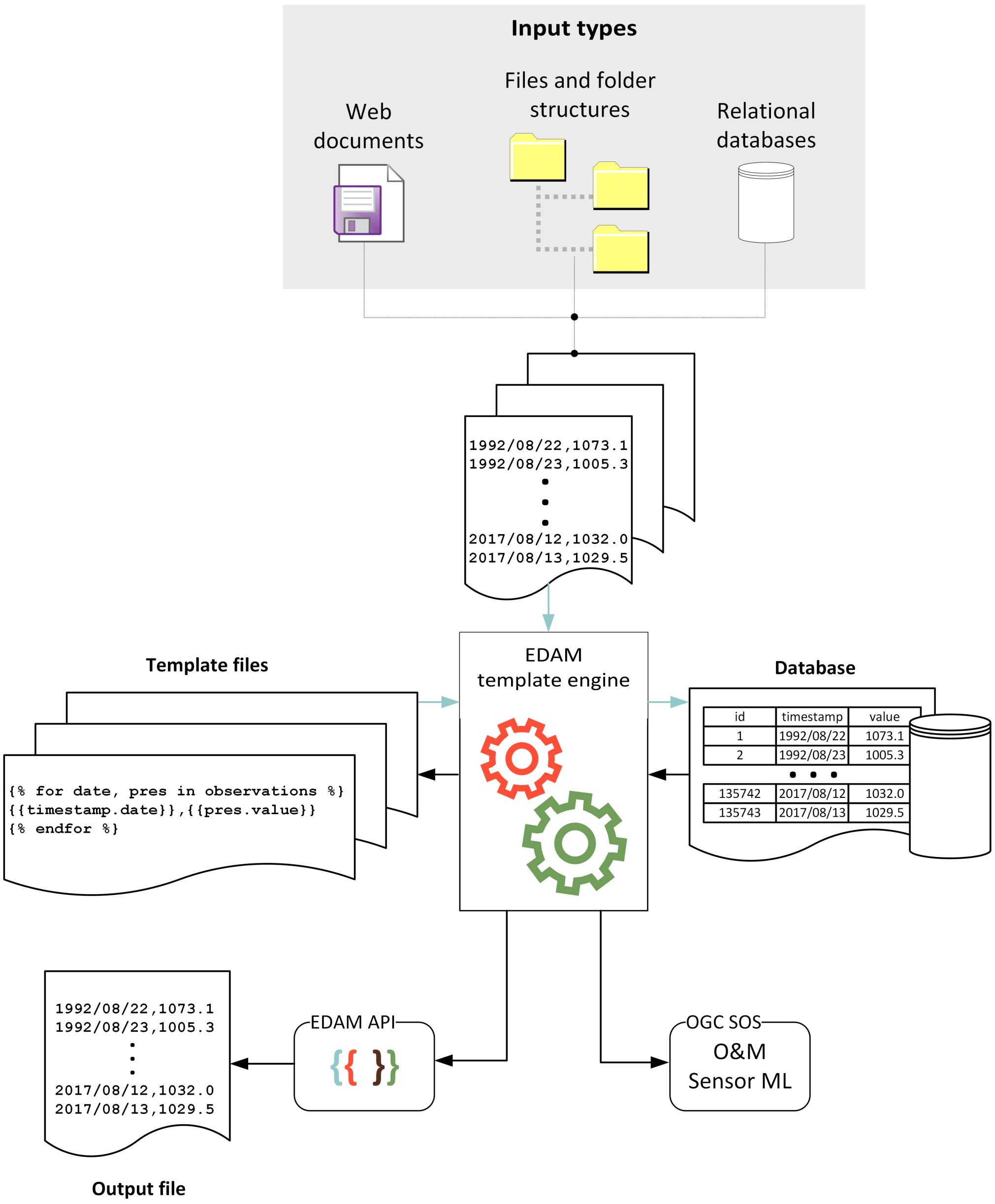


Figure 4.1: *EDAM* abstract architectural design. Black and blue arrows depict output and input workflows, respectively. *EDAM* supports data transformation through its API, and standardized data dissemination through OGC SOS. For the depicted example, input and output files are identical, since the same template file was used for the respecting processes

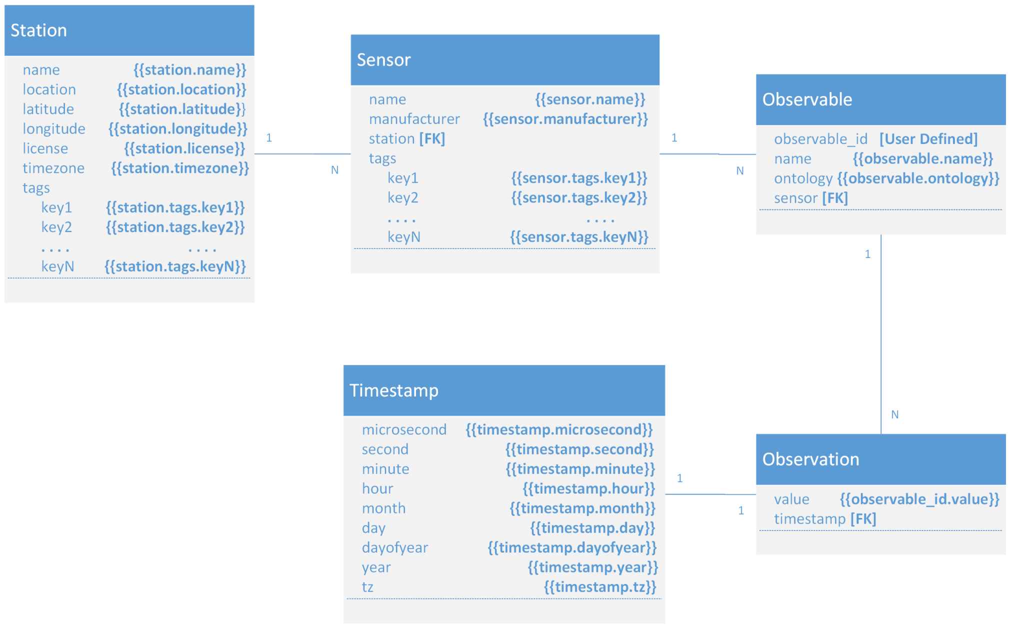

Station
name {{station.name}}
location {{station.location}}
latitude {{station.latitude}}
longitude {{station.longitude}}
license {{station.license}}
timezone {{station.timezone}}
tags
key1 {{station.tags.key1}}
key2 {{station.tags.key2}}
. . . .
keyN {{station.tags.keyN}}
Sensor
name {{sensor.name}}
manufacturer {{sensor.manufacturer}}
station [FK]
tags
key1 {{sensor.tags.key1}}
key2 {{sensor.tags.key2}}
. . . .
keyN {{sensor.tags.keyN}}
Observable
observable_id [User Defined]
name {{observable.name}}
ontology {{observable.ontology}}
sensor [FK]
Timestamp
microsecond {{timestamp.microsecond}}
second {{timestamp.second}}
minute {{timestamp.minute}}
hour {{timestamp.hour}}
month {{timestamp.month}}
day {{timestamp.day}}
dayofyear {{timestamp.dayofyear}}
year {{timestamp.year}}
tz {{timestamp.tz}}
Observation
value {{observable_id.value}}
timestamp [FK]
1
N
1
N
1
N
1
1

Figure 4.2: *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*

(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.

4.3.3 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. Figure 4.3 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,

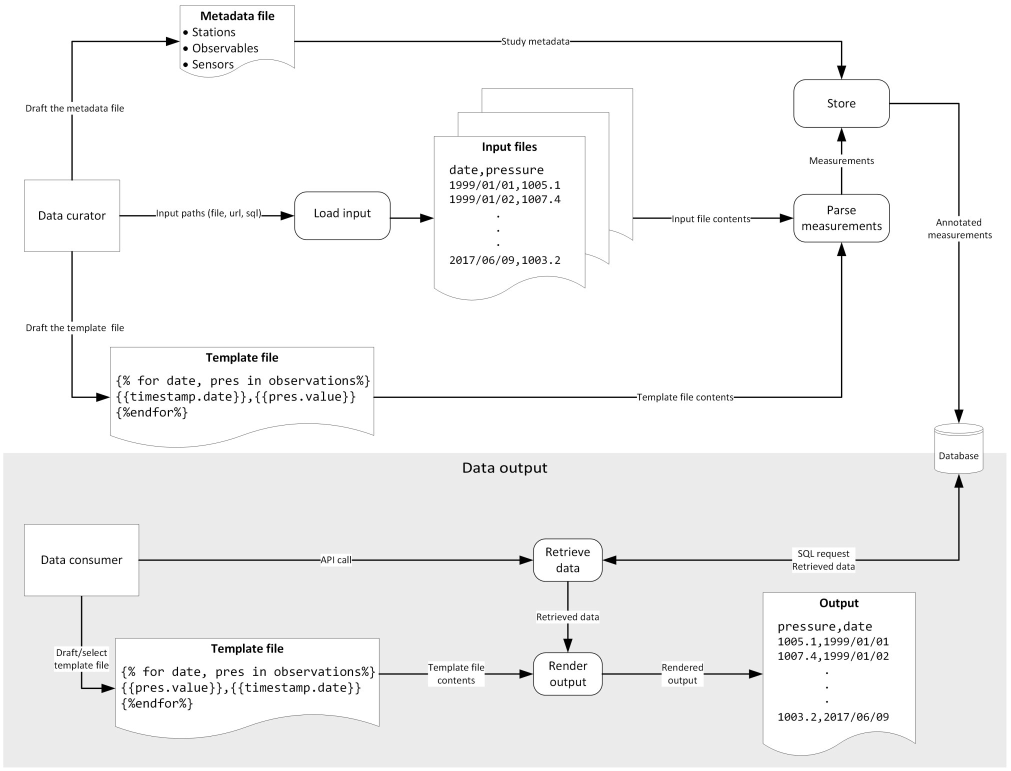


Figure 4.3: *EDAM* workflow. Upper part depicts *data input*, and lower part *data output*. The database component belongs to both. The output template file differs from the input one, as the order of variables is reverted. This is reflected on the *output* data structure

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.

Figure 4.4 shows a sample input file from the UK Met Office (Fig 4.4a), and the corresponding template file (Fig 4.4b). *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

```
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

Figure 4.4: Acquiring meteorological data from a dataset inspired by UK Met Office. (a) depicts the input file and (b) its corresponding template file

`{{tmax.value}}`, `{{tmax.value}}`, `{{tmin.value}}`, `{{af.value}}`, `{{rain.value}}`, and `{{sun.value}}`. Figure 4.5 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, 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.

4.3.4 Implementation and modes of operation

In Table 4.1 we depict *EDAM* implemented functions distinguished 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 on *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 the Flask web framework (Ronacher, 2010), and acquired timeseries are

```
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 4.5: The metadata file for the input dataset reported in Figure 4.4b. 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.

offered as OGC SOS services through the Python implementation reported in (Samourkasidis & 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, 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 4.1: *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

4.4 Demonstration

We demonstrate *EDAM* extended outreach by acquiring environmental timeseries data from diverse data sources. In Table 4.2 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 on text files (HOA) and *relational databases* (BoM (Air)).

In Subsection 4.4.1 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

4.3 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 the following case studies we developed *EDAM* templates as needed and parsed successfully 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 to repeat the experiments with the exception of Swiss TPH and BoM data, as original data involved are not publicly available.

4.4.1 Test cases

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.

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 namedata.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.

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 2017* is:
http://www.bom.gov.au/climate/dwo/**201710**/text/IDCJDW**5002**.**201710**.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.

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.

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.

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.

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.

4.4.2 Demonstrating EDAM output

With regard to dissemination services, an example output of *EDAM* is shown in Figure 4.6. 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{2-8}0{01
-82}.201707.csv" --template bom.tmpl --metadata bom.yaml
```

EDAM processing capabilities are mainly regarded with statistical filters and conditional exports. Figure 4.7 exhibits them when applied on a UK Met dataset. Specifically, we aggregate daily into monthly observations with the `resampling (P1)` function. Consequently, we illustrate `conditional export (P3)` function exporting only those datapoints which satisfy a given condition. *Processing functions* are placed directly on 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 WebX-TREME 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. Figure 4.8 demonstrates the creation of a custom data view, by simply drafting a new template file.

4.4.3 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, 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.

4.5 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-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 4.4 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.

4.5.1 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 among 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.

4.5.2 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.

Table 4.2: 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.

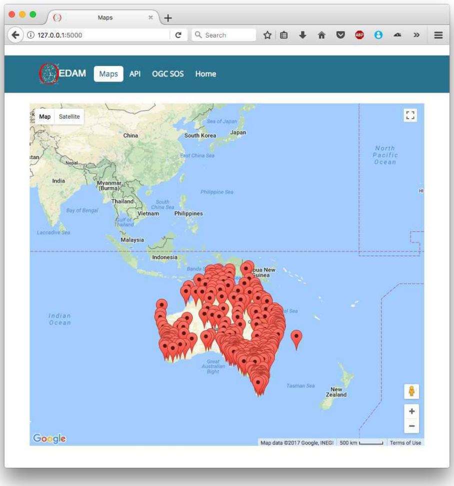
Datasets	**Source**				**Data model**			**Preamble**		**Metadata**	
	file	folder	http	database	tabular	abstract	other	tabular	key-value	included	external
AgMIP	✓				✓						✓
APSIM	✓				✓				✓	✓	✓
BoM (Met)			✓				✓*		✓	✓	✓
UK Met			✓		✓				✓	✓	✓
KNMI	✓				✓			✓		✓	✓
Swiss TPH		✓			✓				✓	✓	✓
HOA			✓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

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

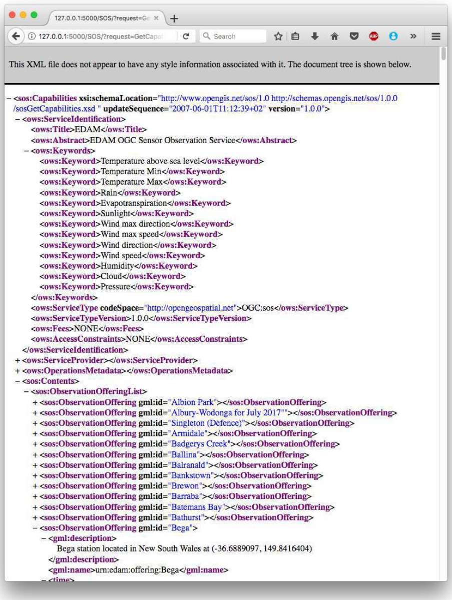
Table 4.3: Input functions utilization for each *EDAM* test case

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

*According to BoM (Met) data model the timestamp of certain data points is projected on their corresponding header column
†The observables of a Swiss TPH station are formatted differently. Thus, parsing is accomplished through more than one *template* files



(a) Sample map projection with EDAM


127.0.0.1:5000/SOS/?request=Ge
127.0.0.1:5000/SOS/?request=GetCapal
Search

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
– <sos:Capabilities xsi:schemaLocation="http://www.opengis.net/sos/1.0 http://schemas.opengis.net/sos/1.0.0
  /sosGetCapabilities.xsd " updateSequence="2007-06-01T11:12:39+02" version="1.0.0">
  – <ows:ServiceIdentification>
      <ows:Title>EDAM</ows:Title>
      <ows:Abstract>EDAM OGC Sensor Observation Service</ows:Abstract>
    – <ows:Keywords>
        <ows:Keyword>Temperature above sea level</ows:Keyword>
        <ows:Keyword>Temperature Min</ows:Keyword>
        <ows:Keyword>Temperature Max</ows:Keyword>
        <ows:Keyword>Rain</ows:Keyword>
        <ows:Keyword>Evapotranspiration</ows:Keyword>
        <ows:Keyword>Sunlight</ows:Keyword>
        <ows:Keyword>Wind max direction</ows:Keyword>
        <ows:Keyword>Wind max speed</ows:Keyword>
        <ows:Keyword>Wind direction</ows:Keyword>
        <ows:Keyword>Wind speed</ows:Keyword>
        <ows:Keyword>Humidity</ows:Keyword>
        <ows:Keyword>Cloud</ows:Keyword>
        <ows:Keyword>Pressure</ows:Keyword>
      </ows:Keywords>
      <ows:ServiceType codeSpace="http://opengeospatial.net">OGC:sos</ows:ServiceType>
      <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
      <ows:Fees>NONE</ows:Fees>
      <ows:AccessConstraints>NONE</ows:AccessConstraints>
    </ows:ServiceIdentification>
  + <ows:ServiceProvider></ows:ServiceProvider>
  + <ows:OperationsMetadata></ows:OperationsMetadata>
  – <sos:Contents>
    – <sos:ObservationOfferingList>
      + <sos:ObservationOffering gml:id="Albion Park"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Albury-Wodonga for July 2017""></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Singleton (Defence)"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Armidale"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Badgerys Creek"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Ballina"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Balranald"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Bankstown"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Brewon"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Barraba"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Batemans Bay"></sos:ObservationOffering>
      + <sos:ObservationOffering gml:id="Bathurst"></sos:ObservationOffering>
      – <sos:ObservationOffering gml:id="Bega">
        – <gml:description>
            Bega station located in New South Wales at (-36.6889097, 149.8416404)
          </gml:description>
          <gml:name>urn:edam:offering:Bega</gml:name>
        – <time>
```

(b) Sample OGC SOS output with EDAM

Figure 4.6: 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

```
{{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

Figure 4.7: 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)

```
@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 4.8: 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)

Chapter 5

A semantic approach for timeseries data fusion

This chapter is based on:

Samourkasidis, A., Athanasiadis, I. A semantic approach for timeseries data fusion [Manuscript submitted for publication]

Abstract

Environmental timeseries acquisition, integration and transformation into a consistent data format is becoming more and more challenging for the Internet of Things produced, but also for legacy model data files. To date, data transformation from diverse sources into one data format requires significant efforts to tackle semantic heterogeneity. In this work we present a declarative approach for environmental timeseries data transformation using semantics. We use a template to annotate environmental data files with terms from a vocabulary. We demonstrate how a reasoner may be employed to resolve synonyms across different vocabularies. This enables to annotate each data file once; and transform its contents using templates with other vocabularies without needing to re-annotate it. We developed a case study where we transform meteorological input files of four agricultural models. With our approach, a certain data file format can be represented through a single template, and by assigning synonym terms we enable automatic transformation into other formats. This facilitates environmental timeseries transformation overcoming semantic heterogeneity, while lowering the e-science barriers.

5.1 Introduction

Scientists and environmental practitioners nowadays are confronted with the vast array of legacy environmental datasets, that become available online, and also with new data produced via the Internet of Things (IoT) devices. Raw data must undergo certain modifications in order for new knowledge to be produced by environmental models (Rizzoli et al., 2007). However, transforming a dataset to be compatible with a certain data specification is a laborious process (Horsburgh et al., 2009) and usually requires a human expert intervention (Athanasiadis, 2015). This process hinders environmental data reusability (Ames et al., 2012), facilitates the formation of data silos (Terrizzano et al., 2015) and ultimately widens the data-to-knowledge gap (Elag et al., 2017).

Semantic heterogeneity among the legacy datasets hinders automatic data transformation. The interdisciplinary nature of environmental sciences impedes reusability which is essential in the era of (big) data (Rizzoli et al., 2007; Wilkinson et al., 2016). Environmental timeseries are typically curated by several organizations and are annotated with implicit semantics (Beran & Piasecki, 2009). The real meaning of the data is obscured in a combination of short data labels or titles combined with institutional knowledge (de Vos et al., 2017). Such implicit semantics concern the physical quantity that was measured (i.e. temperature); the units which were used (i.e. Celcius degrees), and the physical phenomenon (entity or process) that it was measured on (i.e. atmosphere surface air). Often there is implicit knowledge about temporal and spatial references and the observation and measurement protocol. As an example, atmosphere surface air temperature is typically measured with thermometers placed in shelters positioned two meters above ground, according to the World Meteorological Organization (WMO) specifications. Typically unit selection differs among countries, regions and even among different scientific disciplines and domains (Gkoutos et al., 2012). The different ways the observables are quantified with respect to units of measurement adds to the semantic heterogeneity. This can lead to errors in data reuse and interpretation (Horsburgh et al., 2009) and renders data transformation to other formats a rather manual process, which eventually hinders data reuse beyond disciplinary silos.

Utilizing ontologies to support semantic interoperability is not a new concept in the environmental data domain (Gruber & Olsen, 1994). An ontology represents the knowledge of a certain domain in a formalized manner through "a set of statements (axioms) that define concepts and relationships between concepts" (Villa et al., 2009). In the environmental domain, an ontology has been used to identify the physical processes, quantities, and their attributes (e.g. units of measurement) in a standardized manner (Yu & Liu, 2015). There have been several ontologies related to environmental sciences developed in the past decades, which received rather limited adoption (Athanasiadis, 2015). There is also a movement to facilitate data interoperability and reusability (Wilkinson et al., 2016) through the creation of new ontologies and dictionaries, which will be suitable for the Web (Rijgersberg et al., 2011; Compton et al., 2012). However, no clear winner exists among all these ontologies. Thus, several times there needs to be an intermediate step of vocabulary alignment or ontology mediation.

Ontology mediation refers to the process of describing different datasets through one ontology in order a common context to be created and values to be reused (Regueiro et al., 2017). It is used to integrate diverse datasets which each of them is described by a different ontology, in order to become interoperable and reusable (Wilkinson et al., 2016; Shu et al., 2015). The semantic reasoner is a software agent, which infers the implicit relations of an ontology (Mishra & Kumar, 2011), but can also support mediation among a number of them (Bröring et al., 2011c).

In such a diverse ecosystem, lowering the e-science barriers is getting more important than ever (Swain et al., 2016). The declarative semantic approaches which were investigated in the context of integrated environmental modelling (Villa et al., 2009, 2017) offer a significant potential solution to lower this particular barrier.

In this work, we present a declarative approach to cope with semantic heterogeneity in order to automate environmental timeseries processing and transformation. For each data file, we use a template to describe its syntax and a metadata file to annotate the corresponding observables through a vocabulary. Then, a semantic reasoner parses the metadata files and resolves relationships across the different data files. Data stored in a specific format can be automatically transformed to another syntax, with the reasoner inferring compatibility among the corresponding observables. Also, we incorporated a unit of measurement transformation module. We demonstrate this with the weather input files of four crop modelling solutions, namely APSIM (Holzworth et al., 2014), AgMIP (Rosenzweig et al., 2013), DSSAT (Jones et al., 2003), and WOFOST (Diepen et al., 1989) and the meteorological timeseries data provided by the Koninklijk Nederlands Meteorologisch Instituut (KNMI).

The rest of the paper is structured as follows: Section 5.2 reviews contemporary approaches towards environmental data transformation and gives the background of template frameworks. Section 5.3 presents the objectives along with abstract architectural design of our approach and overviews its implementation. Section 5.4 demonstrates the application of the semantic approach and the used datasets. Finally, Section 5.5 discusses our initial key findings, identifies future work and concludes the research.

5.2 Background and related work

The ultimate objective of ontology-driven approaches is the integration of semantically heterogeneous datasets (Villa et al., 2009), i.e. the creation of a consolidated view of datasets that are originally curated by differently, and annotated with different ontologies. This may enable having a single endpoint to submit queries to these heterogeneous datasets (Beran & Piasecki, 2009), providing seamless, frictionless access. In the environmental domain this process is described with many concepts: the terms mediation (Regueiro et al., 2017), translation (Shu et al., 2015) and integration (Leinfelder et al., 2010; Beran & Piasecki, 2009) are synonyms and have been used interchangeably. In the environmental data science literature we discern three approaches towards semantic interoperability, which:

a. built-upon and leverage on approaches which are used to support syntactic interoperability, e.g. environmental data management frameworks such as the ones offered by

Open Geospatial Consortium (OGC) (Bröring et al., 2011a) and CUAHSI (Ames et al., 2012), and spreadsheets,

b. fully utilize the "Semantic Web stack" technologies (e.g. RDF datastores, SPARQL, etc.) (Ziébelin et al., 2017), and
c. utilize scripting[1] to create custom-to-dataset solutions

Usually, the last two approaches cope with both syntactic and semantic heterogeneity at once.

Transformation of syntactically heterogeneous environmental timeseries into a consistent format is the concept around environmental data management frameworks. These frameworks, such the OGC SOS (Bröring et al., 2012) and the CUAHSI HIS (Ames et al., 2012), cope with *syntactic heterogeneity* by hiding the implicit syntaxes of diverse datasets and offering them through consistent data models (e.g. O&M (Cox, 2011), WaterML (Taylor, 2014), etc.). Efforts have also been made towards supporting *semantic* interoperability of such well-established frameworks. Henson et al. (2009) designed a *semantic* extension for the OGC SOS in order to submit high-level queries to raw data. Regueiro et al. (2017) use control vocabularies/ontologies to align different semantics found in distinct data sources. They demonstrate their efforts to construct a semantic mediation version tailored for the OGC SOS needs. Beran and Piasecki developed a knowledge base on top of syntactically interoperable, CUAHSI WaterML formatted datasets. Beran & Piasecki (2009) related terms from local vocabularies which were used to annotate environmental datasets, with terms from a universal ontology. This way, they addressed semantic heterogeneity and provided an endpoint to submit queries to heterogeneous datasets curated by various environmental agencies.

The standardised structure offered by spreadsheets made their utilization popular in the environmental data science domain (de Vos, 2017). This structure accounts for syntactic interoperability, and thus efforts have been made in order to complement those with semantic capabilities. Shu et al. (2015) present their ontology-mediation approach to deal with the *translation* of environmental data encoded in spreadsheets into XML. de Vos et al. (2017) present their ontology mediation approach which concerns the annotation of natural spreadsheets using external vocabularies, in order to identify the domain model implicitly defined in these natural spreadsheets.

Utilizing the Semantic Web stack technologies allows for addressing both syntactic and semantic heterogeneity. The approaches which fall into this category, usually transcribe datasets into semantic-enabled datastores/databases in order to support semantic data linking, processing and querying. Then, they provide a single SPARQL endpoint to perform semantic queries to all underlying datasets (Yu & Liu, 2015). Bizer & Cyganiak (2006) present a tool, called D2R server, which publishes data stored in relational databases to a Semantic Web compatible format. Langegger et al. (2008) describe a mediator-based system for virtual data integration of scientific data. Ziébelin et al. (2017) demonstrate a framework which uses the D2R server (Bizer & Cyganiak, 2006) to semantically link and integrate heterogeneous hydrological data sources. Interestingly, they support for enhanced interoperability as they

[1]We define scripting as the process of creating custom (usually one-off) computer programs to deal with a specific task

disseminate the underlying, integrated datasets through OGC services.

Environmental timeseries integration and transformation via *scripting* have been previously investigated within the agricultural domain. Porter et al. (2014) developed small software programs, called translators to transform the weather data files of four agricultural models into the AgMIP-consistent data format. Similarly, Woodard (2016) in Ag-Analytics developed Python scripts to acquire diverse datasets, store them into a consistent data schema and then offered the transformed data as a service. In both works, the proposed solutions address the syntactic and semantic heterogeneity by aligning all datasets to a consistent data syntax with a predefined data model.

The work presented here is built-upon a mechanism which accounts for syntactic interoperability, and thus falls into the scope of the first approach. Associating a dataset with an abstract representation of its syntax contributes towards syntactic interoperability. Papoutsoglou et al. (2015) introduced the notion of using a template to describe a dataset syntax, parse the corresponding datapoints and offer them as services on the web. In Samourkasidis et al. (2018) we designed and demonstrated a template framework for data acquisition to cope with *syntactic heterogeneity.* Using this framework, e-scientists without a strong computer science background can acquire and reuse environmental timeseries from various outlets (e.g. webpages, local files, databases, etc.) and create custom views of data using templates. In this work we extend this template framework with a declarative approach to cope with semantic heterogeneity.

5.3 Methods

5.3.1 Objectives

There are three objectives in designing and developing a system to support automatic transformation of heterogeneous datasets. The first is to lower the environmental data science barriers, as the target users are e-scientists. As mentioned in Section 5.2, curating environmental datasets is a manual and custom process. In order to cope with semantic heterogeneity and interpret data, users should possess the implicit domain knowledge incorporated in environmental datasets. In this work, we embraced a declarative approach to cope with semantics, which does not require from users more technical skills than those they already have.

The second objective is to support the discovery of compatible datasets. We consider one dataset to be *compatible* with one other, only if the observables reported in the first are *equivalent* with those reported in the other. A semantic reasoner determines *compatibility*, based on semantic annotations provided by users. This enables users to find compatible datasets of interest, originally stored in other formats.

The third objective is *automatic* timeseries transformation between *compatible* formats. The automatic timeseries transformation to different formats consists of two steps: a) syntax transformation, and b) content transformation. The former concerns the layout

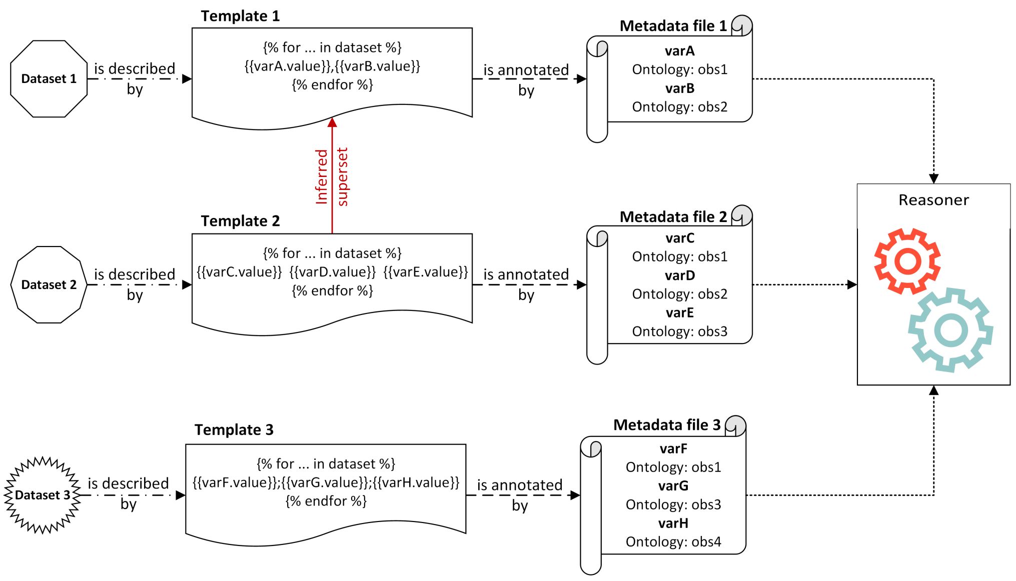


Figure 5.1: The different shapes represent heterogeneous data syntaxes. For each different syntax, the template copes with *syntactic*, and the metadata file with *semantic* heterogeneity. The reasoner infers the compatibility of the data syntaxes based on the ontology definitions declared in the metadata files. The red line arrow depicts an inferred by the reasoner relationship between the templates of two different data syntaxes. *Dataset 2* can be *automatically* transformed to the syntax of the *Dataset 1*. It is a *subset* as it comprises of the same plus some extra observables

transformation, such as the column order. We focused on the latter, that is the unit of measurement transformation of the observables reported in a *source* dataset to match the ones of a *target* dataset. Our approach, will allow to cut across environmental data silos and facilitate timeseries reusability, as it enables users to a) discover datasets in other formats, b) transform them and c) reuse them in their scientific workflows.

5.3.2 Abstract architectural design

There are three key-components involved: a) template files, b) metadata files and c) reasoner. Figure 5.1 depicts the interaction among the components. According to our approach, each distinct data syntax is represented through a template and a metadata file. The reasoner parses the metadata files, stores the ontology definitions for the reported observables in a local ontology and infers compatibility among their corresponding templates. The inferred relationships cope with the semantic heterogeneity, as they support for timeseries transformation among compatible syntaxes.

A *template file* is an abstract representation of a data file contents using programming language agnostic semantics. Users draft one template for each data file *syntax*. They annotate important parts of the dataset using variables. Then, they define the observable metadata, represented through these variables, in a metadata file.

A metadata file is bound to a single template and consists of semantic annotations for the reported observables. Users describe each observable through a name (e.g. Temperature), an ontology class (e.g. ontology:ObservableClass), and if applicable with qualifiers (e.g. max, min, daily). They also provide information about the corresponding units of measurement. For each unit of measurement, the name and symbol are mandatory fields, while a definition through an ontology class is optional. For both observables and units of measurement, users can define equivalent classes from other ontologies.

The reasoner parses the metadata files in order to infer *transformation compatibility* among the templates. Firstly, it creates an instance for each ontology class found in each metadata file. If applicable, it generates on-the-fly concrete subclasses to combine the abstract observable along with its related qualifier(s).
For example, the *maxDailyTemperature* is a *Temperature* subclass which combines a statistical (i.e. max) and a temporal (i.e. daily) qualifier. Secondly, it defines a new class for each template which is described by a general rule, called *axiom*. This axiom asserts in ontology language that the given template comprises of certain observables.

Following compatibility determination comes the unit of measurement transformation. The parser calculates the conversion factor between each set of the compatible *observables*. This calculation is based on the units of measurement which are defined in the source and target metadata files, accordingly. Finally, the conversion factors are applied on-the-fly (if applicable) on each column, and then transformed, according to the target template, in order the dataset instance to be presented to the user.

5.3.3 Use of ontologies

We used a local ontology, which can map concepts and classes defined in different ontologies. This ontology comprises of three high level classes, *Observables*, *Qualifiers*, and *Templates*. In the *Observables* class, we create subclasses for the observables of each dataset, as defined by users in the metadata files. In this version, we annotated observables and units of measurement with classes from a custom, local ontology[2].

The *template variables* which are used to describe the dataset are stored as *instances* of the corresponding *Observables'* subclass. We keep different namespaces for the *instances* of each template. This will enhance findability since each template will have its own prefix. So even for two templates using the same naming for their instances, there will be a distinction among them, based on the used prefixes (e.g. AgMIP:rain and WOFOST:rain). The namespaces can be optionally be defined in the metadata file. In case they are missing they can be generated based on the template file name.

The *Qualifiers* class is further refined into *Statistical* and *Temporal* mutually disjoint subclasses. Based on users' input in metadata files, we define local statistical (e.g. max, min, mean, etc.) and temporal (e.g. daily, hourly, etc.) qualifiers and create their subclasses accordingly.

[2]https://github.com/BigDataWUR/EDAM/blob/master/edam/resources/edam.owl

The *Templates* superclass holds the templates' definitions. We create a subclass for each distinct template along with its axiom definition. The axioms have direct reference to the *Observables* subclasses. The semantic reasoner uses these subclasses, when it comes to inferring compatibility among datasets.

Inferring compatibility among templates is facilitated by this local ontology and its properties. A *hasObservable* object property was defined to establish relationships among the *Templates* classes and their corresponding *Observables.* The axiom of a template with N associated observables defined with the *ontologyA*, is expressed in OWL language as follows:

```
Templates  and (hasObservable some ontologyA:observable1)  and
(hasObservable some ontologyA:observable2) ...  and (hasObservable some
ontologyA:observableN)
```

Based on the template *axioms* the reasoner infers four states of compatibility among two data syntaxes. If A is the *source* and B the *target* template representing different data syntaxes the possible states are:

a. A is **equal** to B, means that both templates comprise of the *same number* of *equivalent* observables.
b. A is a **subset** of B, means template A contains *all equivalent* observables reported in template B, plus one or more additional observables.
c. A is a **superset** of B, is the reversed (b).
d. A is **non-compatible** to B, means that templates A and B may have or not observables in common.

A dataset represented with template A can *automatically* be transformed with template B in the first two cases.

5.3.4 Implementation

This approach utilizes the EDAM template framework Python module reported in (Samourkasidis et al., 2018). It extends the template framework for data acquisition which already copes with syntactic heterogeneity, with a new module to support semantic operations. The system comprises of a parser and a semantic reasoner: EDAM supports the syntax transformation, Owlready2 Python library (Lamy, 2017) the ontology engineering and semantic reasoning, and Pint Python library the units' of measurement trasformation.

We reused open source projects to provide further functionality. Specifically, we developed a parser to extract user definitions about observables and units of measurement from the metadata files, and utilized Owlready2 to store them in a local ontology. Additionally, Owlready2 supports the semantic reasoning to infer compatibility among the semantically heterogeneous datasets. We utilized Pint to support the units' of measurement transformation. Pint calculates the multiplicand factor of two units (i.e. source and target), based on their symbols. By design, Pint supports all SI symbols and their derivatives.

5.3.5 Limitations

The system presented here is intended for environmental timeseries. The system can handle the same file types as EDAM (Samourkasidis et al., 2018), i.e. text-based timeseries stored locally or remotely in one or more files, websites and/or relational databases.

Towards inferring compatibility among datasets, the system takes into consideration only the observables' section in metadata files. The temporal (e.g. hourly, daily, etc) and/or statistical (e.g. min, max, mean, etc.) dimensions of the reported observables should be defined as *qualifiers*. By definition, observables that are reported in different temporal resolutions or regard different statistical value are not compatible. For example, the following sets of source -> target observables are (mutually) incompatible (a, b, c) and compatible (d):

a. *dailyTemperature* -> *dailyMaxTemperature*,
b. *dailyTemperature* -> *hourlyTemperature*,
c. *Temperature* -> *dailyTemperature*,
d. *dailyTemperature* -> *Temperature*,

Users can refer to terms from external ontologies, but these are not directly imported. EDAM creates a local ontology with these terms which serves as a dictionary among the used terms. In this version, external ontologies are not imported to be further used.

The *automatic transformation* refers to the syntax and units' of measurement transformation. Any type of resampling in order source and target dataset temporal resolution to match is not included in the transformation process. Although EDAM offers this service, this is considered as a preprocessing step. Additionally, any possible spatial metadata are not considered when inferring compatibility.

5.4 Demonstration

5.4.1 Case studies

We demonstrate our semantic approach towards environmental timeseries transformation with the weather data files of four environmental models. Table 5.1 presents the selected datasets, the reported observables along with their implicit semantics and units of measurement. Besides the different semantics and units of measurement, each dataset has a different timeseries *syntax*.

For each dataset we developed a template to cope with the diverse *syntaxes* and a metadata file to annotate the reported observables. Figure 5.2 depicts an excerpt of an input dataset for APSIM, the corresponding template (Figure 5.2b), and the metadata file (Figure 5.2c). The variable names (inside the {{}} placeholders) which are used to draft the template, are used in the metadata file to relate observables with their actual meanings. The observables are semantically annotated using a local ontology.

Table 5.1: The *implicit* semantics used by each data syntax to refer to the corresponding observables

Observables	**Datasets**				
	APSIM	AgMIP	DSSAT	WOFOST	KNMI
Solar Radiation	radn (MJ/m^2)	SRAD ($MJ/m2$)	SRAD (MJ/m^2)	irradiation (MJ/m^2)	Q (J/cm^2)
Avg Temperature	-	-	T2M (^oC)	-	TG (d^oC)
Max Temperature	maxt (^oC)	TMAX (^oC)	TMAX (^oC)	maxt (^oC)	TX (d^oC)
Min Temperature	mint (^oC)	TMIN (^oC)	TMIN (^oC)	mint (^oC)	TN (d^oC)
Precipitation	rain (mm)	RAIN (mm)	RAIN (mm)	precip (mm)	RH (dmm)
Wind speed	wind (m/s)	WIND (km/h)	WIND (m/s)	mwind (m/s)	FG (dm/s)
Relative Humidity	RH (%)	RHUM (%)	RH2M (%)	-	UG (%)
Dew Point Temperature	-	DEWP (^oC)	TDEW (^oC)	-	-
Vapor Pressure	-	vprs (kPa)	-	emvp (hPa)	PG ($dhPa$)

```
!!!!   1/01/1961   to   31/12/2005
day    year   radn   maxt   mint   rain   wind   RH
273    2002   17.5   27.2   14.6   0   3.5   54
274    2002   13.6   23.1   14.7   0   5.3   40
275    2002   15.8   27.1   11.1   0   5.5   29
278    2002   15.2   23.1   15.2   0   3.4   47
```

(a) APSIM data file

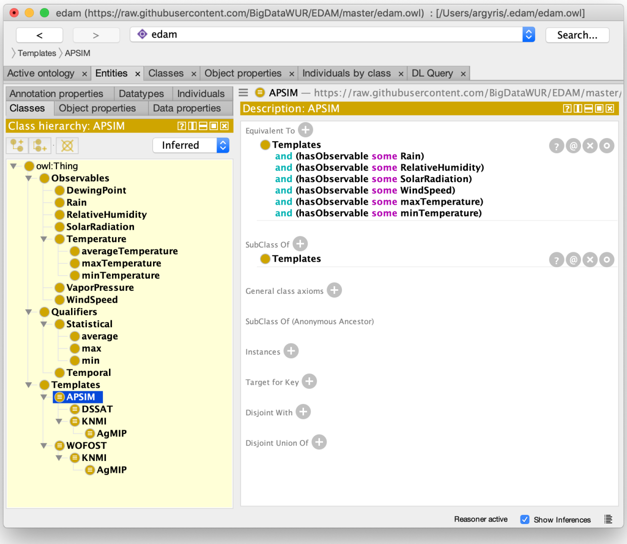
```
!!!!   1/01/1961   to   31/12/2005
day   year   radn   maxt   mint   rain   wind   RH
{%for timestamp, radn,maxt,mint,rain,wind,RH in chunk%}
{{timestamp.dayofyear}} {{timestamp.year}} {{radn.value}} {{maxt.value}}
{{mint.value}} {{rain.value}} {{wind.value}} {{RH.value}}
{%endfor%}
```

(b) APSIM-specific template

```
Observables:
     - observable_id: mint
       name: Temperature
       ontology: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#Temperature
       qualifiers: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#min
     - observable_id: maxt
       name: Max Temperature
       ontology: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#Temperature
       qualifiers: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#max
     - observable_id: rain
       name: Rain
       ontology: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#Rain
     - observable_id: radn
       name: Solar radiation
       ontology: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#SolarRadiation
     - observable_id: wind
       name: Wind
       ontology: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#WindSpeed
     - observable_id: RH
       name: Relative humidity
       ontology: https://github.com/BigDataWUR/EDAM/blob/features/semantics/semedam.owl#RelativeHumidity
Units of Measurement:
     - name: Millijoule per square meters
       symbol: mJ/m^2
       relevant_observables: radn
     - name: Percent
       symbol: \%
       relevant_observables: RH
     - name: Celcius
       symbol: degC
       relevant_observables: mint, maxt
     - name: Millimeters
       symbol: mm
       relevant_observables: rain
     - name: Meters per second
       symbol: m/s
       relevant_observables: wind
```

(c) APSIM-specific metadata file

Figure 5.2: (a) An excerpt from a weather input file inspired by APSIM, (b) the corresponding *template* which represents the APSIM syntax and (c) the *metadata file* which annotates the variables used in (b) with concepts from a local ontology



```
owl:Thing
  Observables
    DewingPoint
    Rain
    RelativeHumidity
    SolarRadiation
    Temperature
      averageTemperature
      maxTemperature
      minTemperature
    VaporPressure
    WindSpeed
  Qualifiers
    Statistical
      average
      max
      min
    Temporal
  Templates
    APSIM
      DSSAT
      KNMI
        AgMIP
    WOFOST
      KNMI
        AgMIP
```

```
Equivalent To
  Templates
    and (hasObservable some Rain)
    and (hasObservable some RelativeHumidity)
    and (hasObservable some SolarRadiation)
    and (hasObservable some WindSpeed)
    and (hasObservable some maxTemperature)
    and (hasObservable some minTemperature)

SubClass Of
  Templates

General class axioms

SubClass Of (Anonymous Ancestor)

Instances

Target for Key

Disjoint With

Disjoint Union Of
```

Figure 5.3: A screenshot of the developed ontology in the Protege software. The *Observables* class consists of the observable types found in the different syntaxes. Combinations of these subclasses, describe each *Template* subclass. The reasoner inferred compatibility as depicted in the class-subclass structure. Specifically, the relationship order is reversed: For example, AgMIP dataset can be automatically transformed according to APSIM, as the first is a subset of the latter

5.4.2 Compatible datasets

The reasoner operated on the five metadata files and updated the local ontology which can be further edited through dedicated ontology editors. It stored a class for each template, and automatically defined the template axiom based on the ontology classes of the related observables.

Figure 5.3 is a screenshot of the Protege ontology editor (Gennari et al., 2003) which depicts the asserted and inferred relationships among the datasets. The compatible datasets are depicted in reverse order. In the class-subclass view depicts the relationships among the datasets. In principle, the subclass dataset is a *subset* of the corresponding class and thus automatic transformation is supported. Based on the template axioms, the reasoner inferred the following relationships:

a. DSSAT *subset* of APSIM,
b. AgMIP *subset* of APSIM,
c. KNMI *subset* of APSIM,
d. KNMI *subset* of WOFOST,
e. AgMIP *subset* of WOFOST

Data compatibility was inferred based on the *combined* observables' classes. These, were generated on-the-fly as a subclass of the respected abstract observable. For example, for the AgMIP `maxt`, the abstract observable is `Temperature` and the statistical qualifier is `max`. This combination results in the on-the-fly generation of `maxTemperature`, which is a `Temperature` subclass.

5.4.3 Automatic transformation

The system automatically transformed the compatible datasets upon user request. Transformation comprises of two parts: the syntax and semantic (or content) transformation. The former was performed by EDAM. The challenge here is regarded with latter: the input and output templates use different semantics (i.e. observable ID). For example, AgMIP and APSIM datasets describe the max Temperature using the `TMAX` and `maxt` identifiers, respectively. The system established a relationship among the underlying observables of the input and output templates based on their compatibility. For example, it inferred that `maxt` and `TMAX` are synonyms and can be used interchangeably.

Unit transformation is performed on-the-fly upon dataset request. System calculated the required conversion factors between source and a target template units and applied them on the corresponding timeseries. Figure 5.4 depicts a KNMI dataset (Figure 5.4a) transformed according to the APSIM format (Figure 5.4b). For this example, the conversion factors for following units' of measurement transformation were calculated and applied on the source dataset:

a. J/cm^2 -> MJ/m^2
b. d°C -> °C
c. dmm -> mm
d. dm/s -> m/s

The system implementation is able to handle incompatible transformation requests, and annotations with unresolvable units of measurement. When a non-compatible transformation is attempted system issued an error. This error informed the user about the (in)compatibility of the involved datasets. The system can also handle units of measurement that either are not expressed correctly or are not SI units. In both cases, system set the conversion factor to 1 (i.e. no transformation) and raised warning messages to the user. For example, in this case a frequently found non-SI unit is the percent unit (`%`).

5.4.4 Lessons learned

In order to fully annotate a dataset (i.e. the observables and units of measurement) users usually require more than one ontology. In general, the ontologies have a specific scope and are intended for specific domains. For example, the OM ontology contains concepts about units of measurement.

The statistical and temporal qualifiers change fundamentally the meaning of the observables but yet they are missing from most ontologies. These qualifiers make observables more specific, rendering the reasoning process more robust. Surprisingly though, their definitions were missing from the three ontologies we used (i.e. OM and Sweet2). The on-the-fly generation of concrete subclasses based on the user-defined qualifiers had a major impact on inferring compatibility.

Inconsistency when annotating observables with qualifiers leads to incompatibility. That is, qualifiers should be used either in *all* or in *none* of the datasets involved. From the datasets we used for the case studies in 5.4.1 only the KNMI included statistical and temporal (daily) qualifiers. As a result, the KNMI: `dailyMeanTemperature` was incompatible with the ASPIM: `Temperature`, even though both refer to the very same observables. Thus, for this example we ignored the temporal and spatial metadata/attributes for the KNMI observed data. Determining the temporal qualifiers of an observable based on reported data in the file records is a step that we will consider for future work.

5.5 Discussion and conclusions

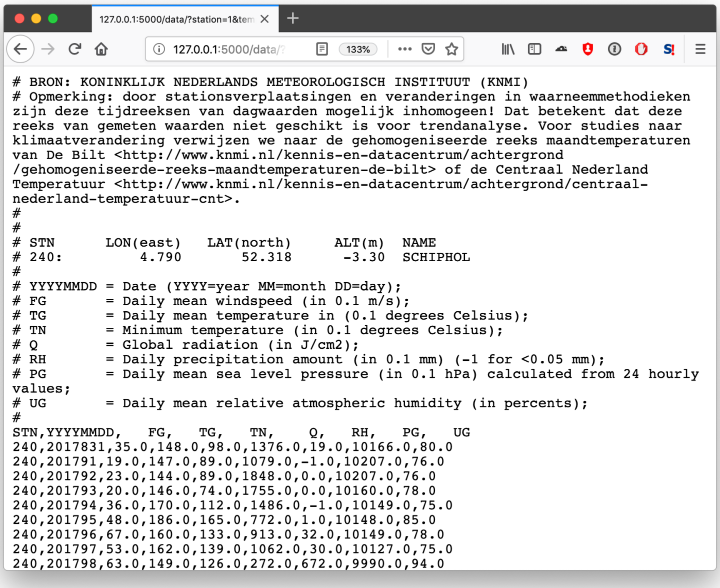
Environmental modelling solutions require their own input types and formats. As these datasets are curated by different entities, there are important differences in terms of *syntax* and *semantics*. Even related modelling solutions, such as APSIM and DSSAT, annotate the same observables through different local vocabularies and sometimes report their observables in different units of measurement (Jones et al., 2017). This *semantic heterogeneity* hinders environmental timeseries reusability, as transforming a dataset to another format is a laborious process (Beran & Piasecki, 2009; Horsburgh et al., 2009) which requires human expert intervention (Athanasiadis, 2015). In this work we present a declarative approach to support environmental timeseries transformation. We employed a reasoner to infer *transformation compatibility* among semantic heterogeneous datasets, and developed a system to support units of measurement transformation. This facilitates the automatic transformation of compatible datasets from one format to another.

The automatic transformation of semantic heterogeneous datasets is essential towards environmental modelling in the IoT era. It cuts across environmental data silos, as it enables timeseries interoperability and reusability (Wilkinson et al., 2016). Users can annotate datasets through a vocabulary, employ the reasoner and transform them into other compatible formats. Additionally, the automatic unit transformation supports e-scientists, as this manual process is often erroneous (Horsburgh et al., 2009). We also consider that this approach contributes towards lowering the e-science barriers (Swain et al., 2016). The proposed declarative approach copes with semantic heterogeneity, and enables e-scientists to transform compatible datasets to a given format, without developing scripts or being ontology engineers themselves.

This work has an exploratory character and sets the groundwork for future work. In this proof of concept our approach supports semantic mediation, by enabling users to annotate the observables of the various datasets with terms for a local ontology. A possible direction for future work may be the design of *intermediate, semantic model-templates*. These model-templates would derive missing observables combining present ones. This is an essential step in cases where two syntaxes are not compatible because of a missing observable. An example from the case studies presented here is the incompatibility of the WOFOST and DSSAT datasets, because of the *Vapor Pressure* observable reported only in the former. A intermediate model-template to derive it, combining the *Temperature* and *Dew point* (present in the DSSAT) would allow automatic transformation between them.

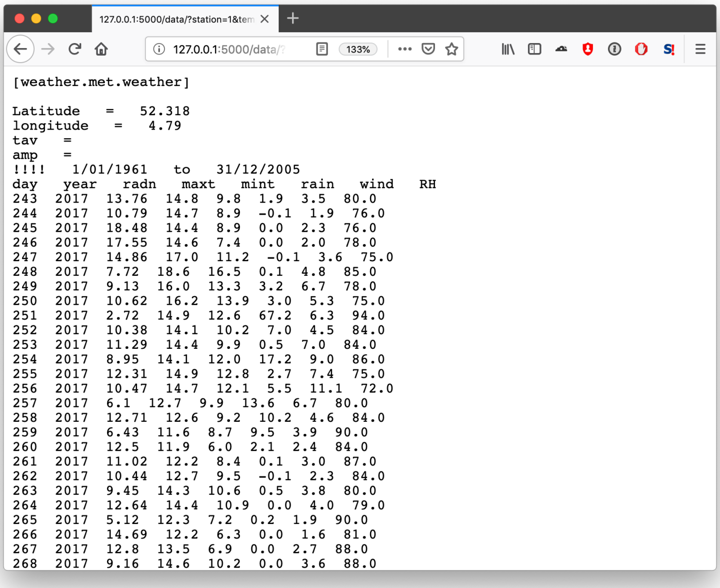
5.5.1 Conclusions

In this work we presented our approach to cope with semantic heterogeneity towards transforming environmental timeseries. We extended a data acquisition template framework which accounts for syntactic interoperability with a reasoner and a unit transformation module. This declarative approach enables users to annotate a data syntax once using terms from a vocabulary and then transform it to other compatible syntaxes. The employed reasoner infers the compatibility among different syntaxes, by creating a semantic description of each one. Then, the unit transformation module determines the relationship among the units and performs on-the-fly transformation where applicable. We demonstrated our declarative approach with the weather input files from four agricultural models and the meterological timeseries data from the Dutch Meteorological Office. In all cases where the reasoner inferred compatibility between two distinct datasets, we were able to transform the syntax and the content of one to another.



```
# BRON: KONINKLIJK NEDERLANDS METEOROLOGISCH INSTITUUT (KNMI)
# Opmerking: door stationsverplaatsingen en veranderingen in waarneemmethodieken
zijn deze tijdreeksen van dagwaarden mogelijk inhomogeen! Dat betekent dat deze
reeks van gemeten waarden niet geschikt is voor trendanalyse. Voor studies naar
klimaatverandering verwijzen we naar de gehomogeniseerde reeks maandtemperaturen
van De Bilt <http://www.knmi.nl/kennis-en-datacentrum/achtergrond
/gehomogeniseerde-reeks-maandtemperaturen-de-bilt> of de Centraal Nederland
Temperatuur <http://www.knmi.nl/kennis-en-datacentrum/achtergrond/centraal-
nederland-temperatuur-cnt>.
#
#
# STN      LON(east)   LAT(north)     ALT(m)  NAME
# 240:         4.790       52.318      -3.30  SCHIPHOL
#
# YYYYMMDD = Date (YYYY=year MM=month DD=day);
# FG       = Daily mean windspeed (in 0.1 m/s);
# TG       = Daily mean temperature in (0.1 degrees Celsius);
# TN       = Minimum temperature (in 0.1 degrees Celsius);
# Q        = Global radiation (in J/cm2);
# RH       = Daily precipitation amount (in 0.1 mm) (-1 for <0.05 mm);
# PG       = Daily mean sea level pressure (in 0.1 hPa) calculated from 24 hourly
values;
# UG       = Daily mean relative atmospheric humidity (in percents);
#
STN,YYYYMMDD,   FG,   TG,   TN,    Q,   RH,   PG,   UG
240,2017831,35.0,148.0,98.0,1376.0,19.0,10166.0,80.0
240,201791,19.0,147.0,89.0,1079.0,-1.0,10207.0,76.0
240,201792,23.0,144.0,89.0,1848.0,0.0,10207.0,76.0
240,201793,20.0,146.0,74.0,1755.0,0.0,10160.0,78.0
240,201794,36.0,170.0,112.0,1486.0,-1.0,10149.0,75.0
240,201795,48.0,186.0,165.0,772.0,1.0,10148.0,85.0
240,201796,67.0,160.0,133.0,913.0,32.0,10149.0,78.0
240,201797,53.0,162.0,139.0,1062.0,30.0,10127.0,75.0
240,201798,63.0,149.0,126.0,272.0,672.0,9990.0,94.0
```

(a) KNMI formatted dataset



```
[weather.met.weather]

Latitude   =   52.318
longitude   =   4.79
tav   =
amp   =
!!!!   1/01/1961   to   31/12/2005
day   year   radn   maxt   mint   rain   wind   RH
243  2017  13.76  14.8  9.8  1.9  3.5  80.0
244  2017  10.79  14.7  8.9  -0.1  1.9  76.0
245  2017  18.48  14.4  8.9  0.0  2.3  76.0
246  2017  17.55  14.6  7.4  0.0  2.0  78.0
247  2017  14.86  17.0  11.2  -0.1  3.6  75.0
248  2017  7.72  18.6  16.5  0.1  4.8  85.0
249  2017  9.13  16.0  13.3  3.2  6.7  78.0
250  2017  10.62  16.2  13.9  3.0  5.3  75.0
251  2017  2.72  14.9  12.6  67.2  6.3  94.0
252  2017  10.38  14.1  10.2  7.0  4.5  84.0
253  2017  11.29  14.4  9.9  0.5  7.0  84.0
254  2017  8.95  14.1  12.0  17.2  9.0  86.0
255  2017  12.31  14.9  12.8  2.7  7.4  75.0
256  2017  10.47  14.7  12.1  5.5  11.1  72.0
257  2017  6.1  12.7  9.9  13.6  6.7  80.0
258  2017  12.71  12.6  9.2  10.2  4.6  84.0
259  2017  6.43  11.6  8.7  9.5  3.9  90.0
260  2017  12.5  11.9  6.0  2.1  2.4  84.0
261  2017  11.02  12.2  8.4  0.1  3.0  87.0
262  2017  10.44  12.7  9.5  -0.1  2.3  84.0
263  2017  9.45  14.3  10.6  0.5  3.8  80.0
264  2017  12.64  14.4  10.9  0.0  4.0  79.0
265  2017  5.12  12.3  7.2  0.2  1.9  90.0
266  2017  14.69  12.2  6.3  0.0  1.6  81.0
267  2017  12.8  13.5  6.9  0.0  2.7  88.0
268  2017  9.16  14.6  10.2  0.0  3.6  88.0
```

(b) KNMI dataset formatted according to APSIM

Figure 5.4: The reasoner inferred compatibility between KNMI and APSIM and dataset depicted in (a) was automatically transformed according to APSIM format (b). The units of measurement transformation was performed on-the-fly

Chapter 6

Synthesis

6.1 Main findings

This thesis investigated the impact of IoT on environmental timeseries data lifecycle from the perspective of e-scientists. In order to address this objective, we refined it into sub-objectives, formulated as research questions. The purpose of this chapter is to revisit them, reflect on the research findings and propose the future work. The sub-objectives, formulated as Research Questions (RQs) are the followings:

a. RQ1: Can environmental timeseries lifecycle be facilitated by IoT prototyping devices?
b. RQ2: Are environmental data dissemination protocols IoT-ready?
c. RQ3: How can e-scientists *acquire*, *integrate* and *transform* timeseries datasets in the heterogeneous IoT ecosystem?

RQ1: Can environmental timeseries lifecycle be facilitated by IoT prototyping devices?

In order to investigate RQ1 we focused on the Raspberry Pi as an IoT prototyping device. The factors which led our decision are regarded with a) Raspberry Pi wide adoption as an IoT enabler device (Johnston & Cox, 2017) and b) the regular hardware updates and Open hardware principles upon which Raspberry Pi is built. In Chapter 2, we performed our assessments on Raspberry Pi 1 Model B. Since then, five new major Raspberry Pi models emerged, each of which offering improvements in hardware specifications, in terms of CPU, GPU and RAM available. Interestingly, the latest models (i.e. Raspberry Pi 3 Model B and B+) feature an embedded, low-power Bluetooth module which is intended for IoT prototyping (Wikipedia contributors, 2019). Finally, Raspberry Pi is increasingly being used in scientific and citizen science endeavors (Johnston & Cox, 2017).

The Raspberry Pi can support persistent data storage, operating as an IoT gateway. In Chapter 2 we investigated the Raspberry Pi, focusing on the data *storage* capabilities and resilience under a suboptimal enabling environment, by simulating power and network outages taking place irregularly. Our 4-month experiment demonstrates that it can facilitate environmental timeseries lifecycle, as it offers persistent data storage under an adverse, IoT-like, enabling environment. This experiment also highlights the Raspberry Pi resilience, as it managed to self-recover after the power and/or network outages.

Despite its low acquisition cost, Raspberry Pi is a multi-purpose device with enough processing power to facilitate environmental timeseries lifecycle processes. In Chapter 2, we evaluated its processing capabilities: We demonstrated that when Raspberry Pi is accompanied with appropriate software it can simultaneously support more processes, next to persistent data storage. Specifically it was able to: a) interface with external sensors to collect measurements, b) further process them to create visualizations, and c) disseminate them with established protocols, such as the OGC SOS. The stress test we conducted revealed that Raspberry Pi was capable to acquire, process and store observations derived from at least four attached sensors, at regular time intervals as low as five seconds, while simultaneously disseminating observations to external users.

RQ2: Are environmental data dissemination protocols IoT-ready?

In order to investigate RQ2, we focused on OGC SOS as an environmental data dissemination protocol. The factors which led our decision are regarded with a) SOS wide adoption in the environmental domain (e.g. hydrology, meteorology, etc.), and b) the existing methodologies (e.g. Mediator/Wrapper architecture (Regueiro et al., 2017)) which support the integration of SOS-disseminated data into scientific workflows.

OGC SOS was not originally intended for the Internet of the Things era. In Chapter 3 we evaluated OGC SOS, and argued that is not suited to operate in an IoT environment, that may involve limited resources and intermittent internet access. We demonstrated that current design of OGC SOS 2.0 is inefficient, as it spares valuable processing power and bandwidth in order to handle invalid user requests. We also proved that OGC SOS lacks the ability to handle disruptions, such as intermittent internet connectivity, as it does not support for progressive data transmission.

We proposed a non-invasive extension to OGC SOS, to make it an IoT-ready dissemination outlet. We designed a pagination extension to support for progressive data transmission. We demonstrated that the pagination extension economizes resources and tackles with limited or interrupted Internet connectivity; rendering OGC SOS to be disruption-tolerant. We conducted a series of tests to verify experimentally the performance improvements offered by our extension. We consider the pagination extension to be backwards-compatible, since it requires minor changes in order to extend legacy SOS servers and clients to implement it. Our pagination extension supports the progressive data transmission, which in turn enables the OGC SOS protocol to become IoT ready.

RQ3: How can e-scientists *acquire*, *integrate* and *transform* timeseries datasets in the heterogeneous IoT ecosystem?

In order to investigate RQ3 we focused on declarative approaches. In Chapters 4 and 5 we reviewed how e-scientists can acquire, integrate and transform heterogeneous timeseries using a) environmental data management frameworks, b) scripting, and c) semantic web technologies (e.g. linked data, ontology engineering, etc.). There are several challenges associated with each methodology. For example, for a) and c) there is a steep learning curve in order to be applied, whereas the scripting methodology provides great flexibility at the price of simplicity. The limitations of the existing methodologies motivated us to investigate the use of a declarative approach to support e-scientists towards acquiring, transforming and integrating heterogeneous datasets.

Declarative approaches can support e-scientists with data curation processes. In Chapter 4 we presented a declarative approach according to which e-scientists can acquire and transform environmental datasets by describing their syntax. The novelty of this approach is that we are not trying to propose another standard, but rather we have developed a domain specific language for describing data file structures in a generic way, using templates. Our approach enables users not only to acquire datasets, but also create custom views, by defining the desired output format again as a template using the same language. This way, e-scientists can re-use environmental timeseries further by feeding them as input in their scientific workflows. Our approach, also supports to automatically expose datasets as services on the web, i.e. through OGC SOS and APIs. We argue that this abstract description of a dataset's syntax with templates is generic and offers a compromise between the generality and complexity tradeoff for syntactic interoperability. We demonstrate the generality of our approach by acquiring timeseries stored in relational databases, files in folder structures, and webpages, in seven case studies from meteorology, agronomy and hydrology.

Semantics are essential towards the integration of datasets acquired from different sources. In Chapter 5 we present an extension to the declarative approach presented in Chapter 4 to cope with semantic heterogeneity towards transforming environmental timeseries. This extension enables e-scientists to annotate the semantics of a data syntax once using terms from a vocabulary. The employed reasoner creates an abstract semantic description of each distinct data syntax, and infers using Description Logic the compatibility among them. We also designed and implemented a unit transformation module, which supports the on-the-fly transformations of compatible measurements when those are reported with different units. We demonstrated the declarative approach transforming the syntax and the content of datasets that are inferred by the reasoner to be compatible.

6.2 The impact of IoT on environmental timeseries lifecycle

The insights and methodologies presented in this thesis concern the environmental e-scientists and practitioners in two ways. First, our contributions in the area of IoT computing requirements and their impact on storage (Chapter 2) and dissemination (Chapter 3) can support e-scientists in order to perform rapid prototyping, hypotheses investigation or to deploy low-cost environmental monitoring campaigns using IoT prototyping devices, such as the Raspberry Pi. We have demonstrated that data storage and dissemination can be performed on-site using IoT prototyping devices and an IoT-ready, backwards-compatible OGC SOS extension, which in turn contributes towards the open-data movement. Second, this thesis findings support e-scientists to use legacy and IoT-produced environmental datasets into scientific workflows and facilitate for data acquisition and integration, by treating syntactic (Chapter 4) and semantic (Chapter 5) heterogeneity of environmental timeseries data.

In Chapter 2, we presented the limitations and capabilities of a low-cost IoT prototyping device. The *computing requirements* (Good et al., 2017) can be lowered, since an IoT prototyping device can have a central role in the environmental timeseries lifecycle. An IoT prototyping device not only can provide persistent data storage and processing capabilities on-site, but it can also support the dissemination of the collected data via a standardized protocol, such as the OGC SOS. In Chapter 3 we argued that the current OGC SOS design is not suitable to operate in an IoT ecosystem. We introduced a backwards-compatible OGC SOS extension that is fit for IoT purposes, and can facilitate the direct integration of the environmental timeseries with existing scientific workflows, as for example in (Regueiro et al., 2015, 2016, 2017).

The active participation of low-cost devices in the environmental data lifecycle processes comes along with the future technological trends. Gartner, Inc. (2018) identifies the IoT-driven edge computing as one of the top technology trends for the future. With the term "edge computing" they refer to "the computing topology in which information processing, and content collection and delivery" (i.e. data dissemination) take place on the IoT devices, close to where the data is being generated. Given the limitations that we investigated only one device, we conclude that IoT gateway devices are able to facilitate several processes in the environmental timeseries lifecycle. This is further supported by the fact that in the meantime, IoT devices have become even more powerful (Gartner, Inc., 2018). It also reveals the potential of the edge computing paradigm for environmental applications. This thesis made contributions towards the realization of future edge computing, environmental science applications by

a. providing with insights about the limits of contemporary IoT gateways and their performance as active participants in the environmental data lifecycle (Chapter 2), and
b. developing an IoT-ready, backwards compatible extension for the OGC SOS to support interoperable data dissemination on-site (Chapter 3).

There is latency in the adoption of new dissemination protocols. For example, OGC SOS 1.0 was introduced in 2007 (Na & Priest, 2007), the updated SOS 2.0 in 2012 (Bröring et al., 2012) and EEA has adopted its second version around 2012 (Jirka et al., 2012). In order to address the IoT challenges, such as the restrained resources of the IoT ecosystem, OGC introduced the SensorThings API in 2016 (Liang et al., 2016). While its performance outpaces SOS and uses all the IoT best practices (e.g. JSON, REST, pagination) it is still under experimentation and not yet adopted by environmental agencies. Interestingly, latency is also observed when organizations need to maintain data infrastructures, as in the case of upgrading a data dissemination protocol to a newer version. For example, NOAA (Center for Operational Oceanographic Products and Services (CO-OPS), 2019) was still operating a custom SOS 1.0 version, when this thesis was written, despite OGC has upgraded SOS to version 2.0 in 2012.

We also note that IoT-produced datasets become "instant-legacy". This is mostly attributed to the plethora of sensing instruments and the lack of data standardization formats. IoT-generated data require manual labour and domain expertise in order to be discovered, interpreted and integrated into applications such as scientific workflows. Surprisingly enough, these are key qualities of legacy data sources. Thus, there is also the danger for IoT-generated, non-standardized data streams to become vanished in the IoT data deluge.

The complexity of the environmental domain hinders smooth data acquisition and integration, even for simple datasets. In Chapters 4 and 5 we focused on meteorological timeseries data, which are less complicated than other data types (e.g. remote sensing, soil data, agromanagement, etc.). However, the plethora of available standards does not contribute directly towards interoperability and create challenges towards their curation. Nowadays, environmental datasets are stored in files, databases, webpages, and are formatted using diverse syntaxes, and can be acquired through standardized protocols (e.g. OGC SOS, Sensor Things API, FIWARE etc.) or via custom APIs. During the EDAM software development (Chapter 4), we discovered a number of issues that led to design changes. These issues validate the complexity of the environmental domain as most of them were regarded with custom data syntaxes and non-standardized temporal and spatial references.

Declarative approaches provide flexibility and can facilitate environmental e-scientists who are not computer science experts. This can be achieved, as declarative approaches hide the underlined complexity, while offer abstract tools to e-scientists for interacting with data. In Chapter 4 we demonstrated how an abstract representation of a dataset syntax along with a template framework can hide the complexity of acquiring and transforming syntactically heterogeneous datasets which are disseminated by different standards or custom APIs. Additionally, as we demonstrated in Chapter 5, a declarative approach can also provide a simple wrapper, hiding complex topics, such as ontology engineering. We argue that declarative approaches contribute towards lowering the *technological skills* requirement, as they assist e-scientists to cope with heterogeneity and other challenges towards timeseries acquisition, transformation and integration.

We were not the first who utilized a declarative approach, but we followed a different path compared to others. The declarative semantic approaches which were investigated in the context of integrated environmental modelling (Villa et al., 2009, 2017) was the inspiration for our approach in Chapter 5. Our approach is differentiated though, as it does not rely upon universal semantics, but enables resources to be annotated with semantics from local vocabularies/ontologies, in a bottom-up approach. This way, data can be represented and interpreted subjectively, according to the vocabulary/ontology used for annotation. Our approach supports the utilization of more that one global ontologies, enabling e-scientists to combine concepts from different domains. In our approach, a local ontology which captures the mappings among concepts of different ontologies, is generated on-the-fly. This is an essential feature, as ontologies have usually a narrow scope and combining them provides more flexibility in annotation.

The fragmentation of the different environmental domains is reflected on the available data formats and semantics. Not only diverse syntaxes (Chapter 4), but also domain specific semantics (Chapter 5) are utilized by the different environmental domains and modelling solutions. These create silo effects among the different environmental domains which have an impact on e-scientists. They introduce additional requirements for them, as besides the manual labor and domain expertise which is required in order to interpret and reuse environmental dataset, they should possess other skills, such as computer science and ontology engineering expertise.

6.3 Directions for future research

This thesis sets the groundwork for future work. We identified four possible directions, which are regarded with:

a. investigating other open IoT challenges, such as privacy and security, in the environmental timeseries lifecycle context,
b. performing and standardizing QA/QC on citizen science produced datasets,
c. extending the use of declarative approaches for more complex operations, and

d. validating the applicability of the approaches and insights presented in this thesis on other data types such as remote sensing, UAV images and others.

Privacy and security aspects should be investigated in the context of environmental timeseries lifecycle. These two IoT challenges are becoming important in the future we envision in this thesis, where IoT gateways play a central role in the environmental data lifecycle processes. The environmental data lifecycle in the edge computing era will involve IoT gateways that facilitate storage and dissemination on behalf of more than one IoT devices, which may belong to different owners. Privacy aspects may be investigated from two perspectives: a) legal (i.e. who owns the data stored), b) technical (i.e. how to implement privacy-preserving mechanisms on an application level). Currently, privacy is getting more attention in the FAIR context (Wilkinson et al., 2016), and there are some applications related to the health domain (Sun et al., 2018), and privacy-preserving citizen campaigns (Drosatos et al., 2014). Future research may also be directed in security aspects, as the IoT devices and gateways have restrained resources and thus are more susceptible to cyber attacks. For example, the current OGC SOS 2.0 is vulnerable to a Denial of Service attack. An attacker can initiate a number of data request occupying the available bandwidth and processing power resources, which are already scarce.

Another research direction may focus on quality assurance and quality control (QA/QC), which is often overlooked in citizen science campaigns. While IoT devices generate a lot of data, these are not often used in scientific workflows as their quality is unknown. We demonstrated that IoT gateways have enough processing power, and thus performing QA/QC processes on-site could be further investigated. Indeed, the efforts towards validating the quality of low-cost sensors' readings are increasing (Gries et al., 2013; Strigaro et al., 2019). However, research efforts should be focused on examining standardized ways to perform and document QA/QC processes. This way, the datasets' quality could be verified in an interoperable manner, and e-scientists can become confident in using citizen science produced data into their scientific workflows.

Declarative approaches could be further investigated towards more complex transformations. One possible direction could be the use of simple, intermediate models executed on the edge. An intermediate model could be defined using a template in order to combine observables (e.g. Temperature and Dew point) to derive new ones (e.g. Vapor Pressure). This extended declarative approach could also be investigated in the context of edge computing, by executing the models on IoT devices. This, could contribute towards economizing bandwidth as only the required, derived measurements can be disseminated. However, there is a tradeoff between the required processing power to calculate the derived observables and the economized bandwidth.

Last but not least, the environmental data lifecycle could be investigated for other data types. The data-intensive IoT produces a number of a data-types beyond timeseries. The approaches and insights presented in this thesis could be validated in other data types, such as those produced by remote sensing, UAVs and cameras. The investigation of those, may provide further insights on how to combine different dataset types in order those to be introduced in composite scientific workflows.

References

Alamdar, F., Kalantari, M., & Rajabifard, A. (2016). Towards multi-agency sensor information integration for disaster management. *Computers, Environment and Urban Systems*, *56*, 68 – 85. doi:10.1016/j.compenvurbsys.2015.11.005.

Alansari, Z., Anuar, N. B., Kamsin, A., Soomro, S., Belgaum, M. R., Miraz, M. H., & Alshaer, J. (2018). Challenges of Internet of Things and Big Data integration. In M. H. Miraz, P. Excell, A. Ware, S. Soomro, & M. Ali (Eds.), *Emerging Technologies in Computing* (pp. 47–55). Springer International Publishing.

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. *Environmental Modelling & Software*, *37*, 146 – 156. doi: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 J. Schweizer, & A. van Herwijnen (Eds.), *Proceedings 1st International Snow Science Workshop (ISSW)*. Davos, Switzerland: Birmensdorf, Swiss Federal Institute for Forest, Snow and Landscape Research WSL.

de Assis, L. F. F., Behnck, L. P., Doering, D., de Freitas, E. P., Pereira, C. E., Horita, F. E., Ueyama, J., & de Albuquerque, J. P. (2016). Dynamic sensor management: Extending Sensor Web for near real-time mobile sensor integration in dynamic scenarios. In *Proceedings Intl. IEEE Advanced Information Networking and Applications (AINA)* (pp. 303–310). doi:10.1109/AINA.2016.100.

Athanasiadis, I., Rizzoli, A., & Beard, D. (2010). Data mining methods for quality assurance in an environmental monitoring network. In *Proceedings 20th Intl Conf on Artificial Neural Networks (ICANN 2010)* (pp. 451–456). Springer Verlag: Thessaloniki, Greece, 2010 volume 6354 of *Lecture Notes in Computer Science*.

Athanasiadis, I. N. (2015). Challenges in modelling of environmental semantics. In *Environmental Software Systems. Infrastructures, Services and Applications* (pp. 19–25). Springer.

Athanasiadis, I. N., Milis, M., Mitkas, P. A., & Michaelides, S. C. (2009). A multi-agent system for meteorological radar data management and decision support. *Environmental Modelling & Software*, *24*, 1264 – 1273. URL: `http://www.sciencedirect.com/science/article/pii/S1364815209001066`. doi:10.1016/j.envsoft.2009.04.010.

Athanasiadis, I. N., & Mitkas, P. A. (2004). An agent-based intelligent environmental monitoring system. *Management of Environmental Quality*, *15*, 238–249.

Athanasiadis, I. N., & Mitkas, P. A. (2007). Knowledge discovery for operational decision support in air quality management. *Journal of Environmental Informatics*, *9*, 100–107.

Atzori, L., Iera, A., & Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, *54*, 2787–2805. doi:10.1016/j.comnet.2010.05.010.

Bahrudin, B., Saifudaullah, M., Abu Kassim, R., & Buniyamin, N. (2013). Development of fire alarm system using Raspberry Pi and Arduino Uno. In *Proceedings Intl. Conference on Electrical, Electronics and System Engineering (ICEESE)* (pp. 43–48). IEEE.

Bayer, M. (2007). Mako templates for python. `http://www.makotemplates.org`. Accessed 1-August-2017.

Bayer, M. (2016). Sqlalchemy: The python sql toolkit and object relational mapper. `http://www.sqlalchemy.org`. URL: `http://www.sqlalchemy.org` accessed 12-December-2016.

Beaujardière, J. D. L. (2016). NOAA environmental data management. *Journal of Map & Geography Libraries*, *12*, 5–27. doi:10.1080/15420353.2015.1087446.

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*, 317–324.

Beran, B., & Piasecki, M. (2009). Engineering new paths to water data. *Computers & Geosciences*, *35*, 753 – 760. URL: `http://www.sciencedirect.com/science/article/pii/S0098300408000988`. doi:10.1016/j.cageo.2008.02.017. Geoscience Knowledge Representation in Cyberinfrastructure.

Bizer, C., & Cyganiak, R. (2006). "D2R" server-publishing relational databases on the semantic web. In *Poster at the 5th international semantic web conference*. volume 175.

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* (pp. 393–410). World Scientific.

Botta, A., de Donato, W., Persico, V., & Pescapé, A. (2016). Integration of Cloud Computing and Internet of Things: a survey. *Future Generation Computer Systems*, *56*, 684–700. doi:10.1016/j.future.2015.09.021.

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

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

Bröering, A., Remke, A., Stasch, C., Autermann, C., Rieke, M., & Möllers, J. (2015). enviroCar: A citizen science platform for analyzing and mapping crowd-sourced car sensor data. *Transactions in GIS*, *19*, 362–376. doi:10.1111/tgis.12155.

Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., & Lemmens, R. (2011a). New Generation Sensor Web Enablement. *Sensors*, *11*, 2652–2699.

doi:10.3390/s110302652.

Bröring, A., Janowicz, K., Stasch, C., Schade, S., Everding, T., & Llaves, A. (2011b). Demonstration: A RESTful SOS Proxy for linked sensor data. In *Proc. 4th Intl. Workshop on Semantic Sensor Networks (SSN11)* (pp. 123–126).

Bröring, A., Maué, P., Janowicz, K., Nüst, D., & Malewski, C. (2011c). Semantically-enabled sensor plug & play for the Sensor Web. *Sensors*, *11*, 7568–7605. URL: `http://www.mdpi.com/1424-8220/11/8/7568`. doi:10.3390/s110807568.

Bröring, 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 1-August-2017.

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., & Schaub, T. (2016). *The GeoJSON Format*. RFC 7946 RFC Editor.

Cagnetti, M., Leccese, F., & Trinca, D. (2013). A new remote and automated control system for the vineyard hail protection based on ZigBee sensors, Raspberry-Pi Electronic Card and WiMAX. *Journal of Agricultural Science and Technology B*, *3*, 853.

Center for Operational Oceanographic Products and Services (CO-OPS) (2019). CO-OPS' Implementation of IOOS Sensor Observation Service (SOS). `https://opendap.co-ops.nos.noaa.gov/ioos-dif-sos/`. Accessed 11-March-2019.

Chang, F.-C., & Huang, H.-C. (2016). A survey on intelligent sensor network and its application. *Journal of Network Intelligence*, *1*, 1–15.

Chiang, M., & Zhang, T. (2016). Fog and IoT: An overview of research opportunities. *IEEE Internet of Things Journal*, *3*, 854–864. doi:10.1109/JIOT.2016.2584538.

Chowdhury, M. N., Nooman, M. S., & Sarker, S. (2013). Access control of door and home security by Raspberry Pi through Internet. *International Journal of Scientific & Engineering Research*, *4*, 550–558.

Compton, M., Barnaghi, P., Bermudez, L., García-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K., Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheth, A., & Taylor, K. (2012). The SSN ontology of the W3C semantic sensor network incubator group. *Journal of Web Semantics*, *17*, 25 – 32. URL: `http://www.sciencedirect.com/science/article/pii/S1570826812000571`. doi:10.1016/j.websem.2012.05.003.

Connors, J. P., Lei, S., & Kelly, M. (2012). Citizen Science in the Age of Neogeography: Utilizing Volunteered Geographic Information for Environmental Monitoring. *Annals of the Association of American Geographers*, *102*, 1267–1289. doi:10.1080/00045608.2011.627058.

Conrad, C. C., & Hilchey, K. G. (2011). A review of citizen science and community-based environmental monitoring: issues and opportunities. *Environmental Monitoring and Assessment*, *176*, 273–291. doi:10.1007/s10661-010-1582-5.

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

Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., & Weerawarana, S. (2002). Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, *6*, 86. doi:10.1109/4236.991449.

Dargie, W., & Poellabauer, C. (2010). *Fundamentals of wireless sensor networks: theory and practice*. Hoboken, NJ, USA: John Wiley & Sons.

Dayan, A., & Hartley, T. (2013). AirPi. `http://airpi.es`. Accessed 12-December-2016.

DCMI Usage Board (2012). Dublin Core Metadata Initiative (DCMI) Metadata Terms. `http://dublincore.org/documents/dcmi-terms/`. Accessed 26-July-2018.

Diepen, C., Wolf, J., Keulen, H., & Rappoldt, C. (1989). WOFOST: a simulation model of crop production. *Soil Use and Management*, *5*, 16–24. doi:10.1111/j.1475-2743.1989.tb00755.x.

Drosatos, G., Efraimidis, P., Athanasiadis, I., Stevens, M., & D'Hondt, E. (2014). Privacy-Preserving Computation of Participatory Noise Maps in the Cloud. *Journal of Systems and Software*, *92*, 170–183. doi:10.1016/j.jss.2014.01.035.

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 International Journal of Geo-Information*, *2*, 553–576. doi: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. *Environmental Modelling & Software*, *94*, 100 – 111. doi:10.1016/j.envsoft.2017.03.032.

Environmental Protection Agency (2016). EPA's Watersheds-based Monitoring and Interoperable Data Platforms Project Lessons Learned. `https://www.epa.gov/sites/production/files/2017-01/documents/iwn_lessonslearned_final_201612.pdf`. Accessed 11-March-2019.

Ferdoush, S., & Li, X. (2014). Wireless sensor network system design using Raspberry Pi and Arduino for environmental monitoring applications. *Procedia Computer Science*, *34*, 103–110.

Foster, I., Katz, D. S., Malik, T., & Fox, P. (2012). Wagging the long tail of earth science: Why we need an earth science data web, and how to build it. `https://pdfs.semanticscholar.org/210d/d61e8f50704401a50376e27338355c9dd61a.pdf`. Accessed 18-September-2019.

Fredericks, J. J., Botts, M., Cook, T., & Bosch, J. (2009). Integrating standards in data QA/QC into OpenGeospatial Consortium sensor observation services. In *OCEANS 2009-EUROPE* (pp. 1–6). doi:10.1109/OCEANSE.2009.5278211.

Free Software Foundation (2016). GNU Affero General Public License. `https://www.gnu.org/licenses/agpl.html`. Accessed 12-April-2019.

Gartner, Inc. (2018). Gartner identifies the top 10 strategic technology trends for 2019. `https://www.gartner.com/en/newsroom/press-releases/2018-10-15-gartner-identifies-the-top-10-strategic-technology-trends-for-2019`. Accessed 13-March-2019.

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 (pp. 49–54). ACM New York, NY, USA: ACM. doi:10.1145/1462802.1462811.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., Noy, N. F., & Tu, S. W. (2003). The evolution of Protege: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, *58*, 89 – 123. URL: `http://www.sciencedirect.com/science/article/pii/S1071581902001271`. doi:10.1016/S1071-5819(02)00127-1.

GeoRSS: (2014). GeoRSS: Geographically Encoded Objects for RSS feeds. `http://www.georss.org`. Accessed 12-December-2016.

Gibert, K., Horsburgh, J. S., Athanasiadis, I. N., & Holmes, G. (2018). Environmental Data Science. *Environmental Modelling & Software*, *106*, 4 – 12. URL: `http://www.sciencedirect.com/science/article/pii/S1364815218301269`. doi:10.1016/j.envsoft.2018.04.005. Special Issue on Environmental Data Science. Applications to Air quality and Water cycle.

Gkoutos, G. V., Schofield, P. N., & Hoehndorf, R. (2012). The units ontology: a tool for integrating units of measurement in science. *Database*, *2012*. doi:10.1093/database/bas033.

Good, N., Ellis, K. A., & Mancarella, P. (2017). Review and classification of barriers and enablers of demand response in the smart grid. *Renewable and Sustainable Energy Reviews*, *72*, 57 – 72. URL: `http://www.sciencedirect.com/science/article/pii/S1364032117300436`. doi:10.1016/j.rser.2017.01.043.

Goodchild, M. F. (2007). Citizens as sensors: the world of volunteered geography. *GeoJournal*, *69*, 211–221. doi:10.1007/s10708-007-9111-y.

Granell, C., Díaz, L., & Gould, M. (2010). Service-oriented applications for environmental models: Reusable geospatial services. *Environmental Modelling & Software*, *25*, 182–198.

Gries, C., Henshaw, D. L., Boose, E. R., Dereszynski, E. W., Shanley, J. B., Taylor, J. R., Porter, J. H., Campbell, J. L., Rustad, L. E., Martin, M. E., & Sheldon, W. M. (2013). Quantity is Nothing without Quality: Automated QA/QC for Streaming Environmental Sensor Data. *BioScience*, *63*, 574–585. doi:10.1525/bio.2013.63.7.10.

Gruber, T. R., & Olsen, G. R. (1994). An Ontology for Engineering Mathematics. In J. Doyle, P. Torasso, & E. Sandewall (Eds.), *Principles of Knowledge Representation and Reasoning* (pp. 258–269). Bonn, Germany: Morgan Kaufmann. doi:10.1016/B978-1-4832-1452-8.50120-2.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, *29*, 1645–1660.

Hart, J. K., & Martinez, K. (2015). Towards an Environmental Internet of Things. *Earth and Space Science*, *2*, 194–200. doi:10.1002/2014EA000044.

Harth, A., Knoblock, C., Stadtmüller, S., Studer, R., & Szekely, P. (2013). On-the-fly integration of static and dynamic sources. In O. Hartig, J. Sequeda, A. Hogan, & T. Matsutsuk (Eds.), *Proceedings 4th International Workshop on Consuming Linked Data (COLD2013)*. CEUR-WS.org volume 1034.

Hartley, T. (2013). AirPi software. `https://github.com/tomhartley/AirPi`. Accessed 12-December-2016.

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, *47*, 98–115. doi:10.1016/j.is.2014.07.006.

Havlik, D., Bleier, T., & Schimak, G. (2009). Sharing Sensor Data with SensorSA and Cascading Sensor Observation Service. *Sensors*, *9*, 5493–5502. URL: `https://dx.doi.org/10.3390/s90705493`. doi:10.3390/s90705493.

Henson, C. A., Pschorr, J. K., Sheth, A. P., & Thirunarayan, K. (2009). Semsos: Semantic sensor observation service. In *Collaborative Technologies and Systems, 2009. CTS'09. International Symposium on* (pp. 44–53). IEEE.

Hey, T., & Trefethen, A. (2003). e-Science and its implications. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, *361*, 1809–1825.

Hey, T., & Trefethen, A. E. (2005). Cyberinfrastructure for e-Science. *Science*, *308*, 817–821. URL: `http://science.sciencemag.org/content/308/5723/817`. doi:10.1126/science.1110410.

Heyman, J., Hamrén, J., Byström, C., & Heyman, H. (2011). Locust: An open source load testing tool. `http://locust.io`. Accessed 12-December-2016.

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. *Environmental Modelling & Software*, *62*, 327 – 350. doi: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. *Environmental Modelling and Software*, *72*, 276–286. doi: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. *Computers & Geosciences*, *80*, 84–94. doi:10.1016/j.cageo.2015.04.001.

Horsburgh, J. S., Aufdenkampe, A. K., Mayorga, E., Lehnert, K. A., Hsu, L., Song, L., Jones, A. S., Damiano, S. G., Tarboton, D. G., Valentine, D., Zaslavsky, I., & Whitenack, T. (2016). Observations Data Model 2: A community information model for spatially discrete Earth observations. *Environmental Modelling & Software*, *79*, 55 –

74. URL: `http://www.sciencedirect.com/science/article/pii/S1364815216300093`. doi:10.1016/j.envsoft.2016.01.010.

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). URL: `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 Resources Research*, *44*. doi:10.1029/2007WR006392.

Horsburgh, J. S., Tarboton, D. G., Maidment, D. R., & Zaslavsky, I. (2011). Components of an environmental observatory information system. *Computers & Geosciences*, *37*, 207 – 218. doi:10.1016/j.cageo.2010.07.003.

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. *Environmental Modelling & Software*, *24*, 879–888. doi:10.1016/j.envsoft.2009.01.002.

IEEE International Conference on eScience (2018). What is eScience? `https://escience-conference.org`. Accessed 26-July-2018.

Ihaka, R., & Gentleman, R. (1996). R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, *5*, 299–314. doi:10.1080/10618600.1996.10474713.

Jankowski, N. W. (2007). Exploring e-Science: An Introduction. *Journal of Computer-Mediated Communication*, *12*, 549–562. doi:10.1111/j.1083-6101.2007.00337.x.

Janowicz, K., Bröring, A., Stasch, C., Schade, S., Everding, T., & Llaves, A. (2013). A RESTful Proxy and Data Model for Linked Sensor Data. *International Journal of Digital Earth*, *6*, 233–254. doi:10.1080/17538947.2011.614698.

Jazayeri, M. A., Huang, C.-Y., & Liang, S. H. L. (2012). TinySOS: Design and Implementation of Interoperable and Tiny Web Service for the Internet of Things. In *Proceedings 1st ACM SIGSPATIAL Workshop on Sensor Web Enablement* SWE '12 (pp. 39–46). New York, NY, USA: ACM. doi:10.1145/2451716.2451722.

Jazayeri, M. A., Liang, S. H., & Huang, C.-Y. (2015). Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things. *Sensors*, *15*, 24343–24373. doi:10.3390/s150924343.

Jirka, S., & Bröring, A. (2012). Practical experiences with Sensor Web technology. In *Proc Sensing a Changing World*.

Jirka, S., Bröring, A., Kjeld, P., Maidens, J., & Wytzisk, A. (2012). A lightweight approach for the Sensor Observation Service to share environmental data across Europe. *Transactions in GIS*, *16*, 293–312. doi:10.1111/j.1467-9671.2012.01324.x.

Johnston, S. J., & Cox, S. J. (2017). The Raspberry Pi: A technology disrupter, and the enabler of dreams. *Electronics*, *6*. URL: `http://www.mdpi.com/2079-9292/6/3/51`. doi:10.3390/electronics6030051.

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. *Environmental monitoring and assessment*, *187*, 1–19. doi:10.1007/s10661-015-4594-3.

Jones, J., Hoogenboom, G., Porter, C., Boote, K., Batchelor, W., Hunt, L., Wilkens, P., Singh, U., Gijsman, A., & Ritchie, J. (2003). The DSSAT cropping system model. *European Journal of Agronomy*, *18*, 235 – 265. URL: `http://www.sciencedirect.com/science/article/pii/S1161030102001077`. doi:10.1016/S1161-0301(02)00107-7. Modelling Cropping Systems: Science, Software and Applications.

Jones, J. W., Antle, J. M., Basso, B., Boote, K. J., Conant, R. T., Foster, I., Godfray, H. C. J., Herrero, M., Howitt, R. E., Janssen, S., Keating, B. A., Munoz-Carpena, R., Porter, C. H., Rosenzweig, C., & Wheeler, T. R. (2017). Brief history of agricultural systems modeling. *Agricultural Systems*, *155*, 240 – 254. URL: `http://www.sciencedirect.com/science/article/pii/S0308521X16301585`. doi:https://doi.org/10.1016/j.agsy.2016.05.014.

jQuery (2016). jquery. `https://jquery.com`. Accessed 12-December-2016.

Jung, M., Weidinger, J., Kastner, W., & Olivieri, A. (2013). Building automation and smart cities: An integration approach based on a service-oriented architecture. In *Proc 27th Intl Conf Advanced Information Networking and Applications Workshops (WAINA)* (pp. 1361–1367). IEEE.

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* (pp. 3363–3372). ACM. doi: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. *European Journal of Agronomy*, *18*, 267 – 288. doi:10.1016/S1161-0301(02)00108-9. Modelling Cropping Systems: Science, Software and Applications.

Khan, R., Khan, S. U., Zaheer, R., & Khan, S. (2012). Future Internet: The Internet of Things architecture, possible applications and key challenges. In *Proceedings 10th Intl. Conference on Frontiers of Information Technology* (pp. 257–260). doi:10.1109/FIT.2012.53.

Kjeld, P., Bliki, J., Jirka, S., & Wytzisk, A. (2011). Sensor web technology for sharing environmental data across Europe. In *Proceedings of the INSPIRE Conference*.

Klein, T., Samourkasidis, A., Athanasiadis, I. N., Bellocchi, G., & Calanca, P. (2017). webxtreme: R-based web tool for calculating agroclimatic indices of extreme events. *Computers and Electronics in Agriculture*, *136*, 111 – 116. doi:10.1016/j.compag.2017.03.002.

Kruger, C. P., & Hancke, G. P. (2014). Benchmarking Internet of Things devices. In *12th Intl. Conf. on Industrial Informatics (INDIN)* (pp. 611–616). IEEE.

Lagoze, C., & Van de Sompel, H. (2001). The open archives initiative: Building a low-barrier interoperability framework. In *Proc 1st ACM/IEEE-CS Joint Conference on Digital libraries* JCDL '01 (pp. 54–62). New York, NY, USA: ACM. doi:10.1145/379437.379449.

Lamy, J.-B. (2017). Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in*

Medicine, *80*, 11 – 28. URL: `http://www.sciencedirect.com/science/article/pii/S0933365717300271`. doi:10.1016/j.artmed.2017.07.002.

Lanfranchi, V., Ireson, N., When, U., Wrigley, S., & Fabio, C. (2014). Citizens' observatories for situation awareness in flooding. In *Proc 11th Intl Conf Information Systems for Crisis Response and Management (ISCRAM)* (pp. 145–154).

Langegger, A., Wöß, W., & Blöchl, M. (2008). A semantic web middleware for virtual data integration on the web. In S. Bechhofer, M. Hauswirth, J. Hoffmann, & M. Koubarakis (Eds.), *The Semantic Web: Research and Applications* (pp. 493–507). Berlin, Heidelberg: Springer Berlin Heidelberg.

Laursen, O., & Schnur, D. (2007). Flot: Attractive JavaScript plotting for jQuery. `http://www.flotcharts.org`. Accessed 12-December-2016.

Leccese, F., Cagnetti, M., & Trinca, D. (2014). A smart city application: A fully controlled street lighting isle based on Raspberry-Pi card, a ZigBee Sensor Network and WiMAX. *Sensors*, *14*, 24408–24424.

Leinfelder, B., Tao, J., Costa, D., Jones, M. B., Servilla, M., O'Brien, M., & Burt, C. (2010). A metadata-driven approach to loading and querying heterogeneous scientific data. *Ecological Informatics*, *5*, 3 – 8. URL: `http://www.sciencedirect.com/science/article/pii/S1574954109000685`. doi:10.1016/j.ecoinf.2009.08.006. Special Issue: Advances in environmental information management.

Lewis, A., Campbell, M., & Stavroulakis, P. (2016). Performance evaluation of a cheap, open source, digital environmental monitor based on the Raspberry Pi. *Measurement*, *87*, 228–235.

Li, S., Da Xu, L., & Zhao, S. (2015). The Internet of Things: a survey. *Information Systems Frontiers*, *17*, 243–259. doi:10.1007/s10796-014-9492-7.

Liang, S., Huang, C., Khalafbeigi, T., Liang, S., Huang, C., & Khalafbeigi, T. (2016). *OGC SensorThings API Part 1: Sensing*. Implementation Standard 15-078r6 Open Geospatial Consortium.

Mahmoud, R., Yousuf, T., Aloul, F., & Zualkernan, I. (2015). Internet of things (IoT) security: Current status, challenges and prospective measures. In *10th Intl. Conf. on Internet Technology and Secured Transactions (ICITST)* (pp. 336–341). IEEE.

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. *Environmental Modelling & Software*, *51*, 59–69.

McFerren, G., Hohls, D., Fleming, G., & Sutton, T. (2009). Evaluating Sensor Observation Service implementations. In *Proceedings Intl. Geoscience and Remote Sensing Symposium (IGARSS)* (pp. 363–366). IEEE volume 5. doi: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)*. New York, NY, USA: ACM.

Miorandi, D., Sicari, S., Pellegrini, F. D., & Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, *10*, 1497 – 1516. URL: `http://www.sciencedirect.com/science/article/pii/S1570870512000674`. doi:10.1016/j.adhoc.2012.02.016.

Mishra, R. B., & Kumar, S. (2011). Semantic web reasoners and languages. *Artificial Intelligence Review*, *35*, 339–368. doi:10.1007/s10462-010-9197-3.

Moure, D., Torres, P., Casas, B., Toma, D., Blanco, M. J., Del Río, J., & Manuel, A. (2015). Use of low-cost acquisition systems with an embedded linux device for volcanic monitoring. *Sensors*, *15*, 20436–20462.

Muller, C., Chapman, L., Johnston, S., Kidd, C., Illingworth, S., Foody, G., Overeem, A., & Leigh, R. (2015). Crowdsourcing for climate and atmospheric sciences: Current status and future potential. *International Journal of Climatology*, *35*, 3185–3203.

Mulligan, G., & Gračanin, D. (2009). A comparison of SOAP and REST implementations of a service based interaction independence middleware framework. In *Proc. Winter Simulation Conference (WSC)* (pp. 1423–1432). doi:10.1109/WSC.2009.5429290.

Na, A., & Priest, M. (2007). *OGC Sensor Observation Service 1.0*. Implementation Standard 06-009r6 Open Geospatial Consortium.

Negru, C., Pop, F., Mocanu, M., & Cristea, V. (2016). A unified approach to data modeling and management in big data era. In Z. Mahmood (Ed.), *Data Science and Big Data Computing* chapter Data Science and Big Data Computing. (pp. 95–116). Springer International Publishing. doi:10.1007/978-3-319-31861-5_5.

Nikhade, S. G. (2015). Wireless sensor network system using Raspberry Pi and ZigBee for environmental monitoring applications. In *Proc Intl Conf Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)* (pp. 376–381). IEEE.

Nuttall, B. (2016). Top 10 Raspberry Pi add-on boards. `https://opensource.com/life/16/7/top-10-Raspberry-Pi-boards`. Accessed 12-December-2016.

Oetiker, T. (2014). RRDtool. `http://oss.oetiker.ch/rrdtool/`. Accessed 12-December-2016.

Papazoglou, M., & Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM*, *46*, 25. doi:10.1145/944217.944233.

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 V. K. Johannsen, S. Jensen, V. Wohlgemuth, C. Preist, & E. Eriksson (Eds.), *Adjunct Proc. 29th EnviroInfo and 3rd ICT4S Conference* (pp. 65–74). University of Copenhagen. ISBN 978-87-7903-712-0.

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. *Computers & Geosciences*, *53*, 154 – 161. doi:10.1016/j.cageo.2012.04.019. Modeling for Environmental Change.

Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Sensing as a service model for smart cities supported by Internet of Things. *Transactions on Emerging Telecommunications Technologies*, *25*, 81–93. doi:10.1002/ett.2704.

Pinto, Y. (2019). Why microSD Express Memory Cards Rule: The Many Advantages of the PCIe and NVMe Standards . `https://www.sdcard.org/press/thoughtleadership/190315_Why_microSD_Express_Memory_Cards_Rule_The_Many_Advantages_of_the_PCIe_and_NVMe_Standards.html`. Accessed 12-April-2019.

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. *Environmental Modelling & Software*, *62*, 495–508. doi:10.1016/j.envsoft.2014.09.004.

Pradilla, J., González, R., Esteve, M., & Palau, C. (2016). Sensor Observation Service (SOS)/Constrained Application Protocol (CoAP) proxy design. In *Proceedings 18th Mediterranean Electrotechnical Conference (MELECON)* (pp. 1–5). doi:10.1109/MELCON.2016.7495411.

Pradilla, J., Palau, C., & Esteve, M. (2015). SOSLITE: Lightweight Sensor Observation Service (SOS) for the Internet of Things (IoT). In *ITU Kaleidoscope: Trust in the Information Society (K-2015)* (pp. 1–7). IEEE. doi:10.1109/Kaleidoscope.2015.7383625.

Python Software Foundation (2016). The Python standard library: sqlite3 – DB-API 2.0 interface for SQLite databases. `https://docs.python.org/2/library/sqlite3.html`. Accessed 26-July-2018.

Python Software Foundation (2017a). The Python standard library: random – Generate pseudo-random numbers. `https://docs.python.org/2/library/random.html`. Accessed 26-July-2018.

Python Software Foundation (2017b). The Python standard library: time – Time access and conversions. `https://docs.python.org/2/library/sqlite3.html`. Accessed 26-July-2018.

Python Software Foundation (2018). PyPI - the Python Package Index. `https://pypi.python.org/pypi`. Accessed 22-February-2018.

Raskin, R. G., & Pan, M. J. (2005). Knowledge representation in the Semantic Web for Earth and Environmental Terminology (SWEET). *Computers & Geosciences*, *31*, 1119–1125.

Raspberry Pi (2018). Raspberry Pi Model B. `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`. Accessed 12-July-2018.

Re, G. L., Peri, D., & Vassallo, S. D. (2014). Urban air quality monitoring using vehicular sensor networks. In *Advances onto the Internet of Things* (pp. 311–323). Springer.

Regueiro, M. A., Viqueira, J. R., Stasch, C., & Taboada, J. A. (2017). Semantic mediation of observation datasets through Sensor Observation Services. *Future Generation Computer Systems*, *67*, 47 – 56. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X16302722`. doi:10.1016/j.future.2016.08.013.

Regueiro, M. A., Viqueira, J. R., Taboada, J. A., & Cotos, J. M. (2015). Virtual integration of sensor observation data. *Computers & Geosciences*, *81*, 12–19.

doi:10.1016/j.cageo.2015.04.006.

Regueiro, M. A., Viqueira, J. R. R., Stasch, C., & Taboada, J. A. (2016). Sensor Observation Service semantic mediation: Generic wrappers for in-situ and remote devices. In I. C.-W. et al. (Ed.), *ER* (pp. 269–276). Springer International Publishing volume 9974 of *LNCS*. doi:10.1007/978-3-319-46397-1 21.

Reitz, K. (2017). Requests: HTTP for humans. `http://docs.python-requests.org/en/master/`. Accessed 22-January-2017.

Richardson, L., & Ruby, S. (2008). *RESTful web services*. Sebastopol, CA, USA: O'Reilly Media, Inc.

Rijgersberg, H., Wigham, M., & Top, J. (2011). How semantics can improve engineering processes: A case of units of measure and quantities. *Advanced Engineering Informatics*, *25*, 276 – 287. URL: `http://www.sciencedirect.com/science/article/pii/S1474034610000753`. doi:10.1016/j.aei.2010.07.008. Information mining and retrieval in design.

Rizzoli, A. E., Athanasiadis, I. N., & Villa, F. (2007). Delivering environmental knowledge: a semantic approach. In *Proc. 21st International Conference on Informatics for Environmental Protection: EnviroInfo* (pp. 43–50).

Ronacher, A. (2008). Jinja2. `http://jinja.pocoo.org`. Accessed 12-December-2016.

Ronacher, A. (2010). Flask. `http://flask.pocoo.org`. Accessed 12-December-2016.

Rosenzweig, C., Jones, J., Hatfield, J., Ruane, A., Boote, K., Thorburn, P., Antle, J., Nelson, G., Porter, C., Janssen, S., Asseng, S., Basso, B., Ewert, F., Wallach, D., Baigorria, G., & Winter, J. (2013). The Agricultural Model Intercomparison and Improvement Project (AgMIP): Protocols and pilot studies. *Agricultural and Forest Meteorology*, *170*, 166 – 182. URL: `http://www.sciencedirect.com/science/article/pii/S0168192312002857`. doi:10.1016/j.agrformet.2012.09.011. Agricultural prediction using climate model ensembles.

Rouached, M., Baccar, S., & Abid, M. (2012). RESTful Sensor Web Enablement Services for Wireless Sensor Networks. In *IEEE Eighth World Congress on Services* (pp. 65–72). IEEE. doi:10.1109/SERVICES.2012.48.

Samourkasidis, A., & Athanasiadis, I. N. (2014). Towards a low-cost, full-service air quality data archival system. In *In Proceedings of the 7th Intl Congress on Environmental Modelling and Software, International Environmental Modelling and Software Society (iEMSs)*.

Samourkasidis, A., & Athanasiadis, I. N. (2016). Airchive software. `https://github.com/ecologismico/airchive`. Accessed 12-December-2016.

Samourkasidis, A., & Athanasiadis, I. N. (2017). A miniature data repository on a Raspberry Pi. *Electronics*, *6*. doi:10.3390/electronics6010001.

Samourkasidis, A., Papoutsoglou, E., & Athanasiadis, I. N. (2018). A template framework for environmental timeseries data acquisition. *Environmental Modelling & Software*, *117*, 237–249. doi:10.1016/j.envsoft.2018.10.009.

Samourkasids, A., Athanasiadis, I. N., & Papoutsoglou, E. (2018). Edam software. `https:`

//github.com/BigDataWUR/EDAM. Accessed 22-February-2018.

Sanchez, L., Muñoz, L., Galache, J. A., Sotres, P., Santana, J. R., Gutierrez, V., Ramdhany, R., Gluhak, A., Krco, S., Theodoridis, E., & Pfisterer, D. (2014). SmartSantander: IoT experimentation over a smart city testbed. *Computer Networks*, *61*, 217 – 238. doi:10.1016/j.bjp.2013.12.020. Special issue on Future Internet Testbeds – Part I.

Santos, P. M., Rodrigues, J. G. P., Cruz, S. B., Lourenço, T., d'Orey, P. M., Luis, Y., Rocha, C., Sousa, S., Crisóstomo, S., Queirós, C., Sargento, S., Aguiar, A., & Barros, J. (2018). PortoLivingLab: An IoT-Based Sensing Platform for Smart Cities. *IEEE Internet of Things Journal*, *5*, 523–532. doi:10.1109/JIOT.2018.2791522.

Sapes, J., & Solsona, F. (2016). FingerScanner: Embedding a fingerprint scanner in a Raspberry Pi. *Sensors*, *16*, 220.

Schön, A., Streit-Juotsa, L., & Schumann-Bölsche, D. (2014). Raspberry Pi and Sensor Networking for African health supply chains. In *Proceedings 6th Intl Conf Operations and Supply Chain Management, Bali*.

Shu, Y., Ratcliffe, D., Compton, M., Squire, G., & Taylor, K. (2015). A semantic approach to data translation: A case study of environmental observations data. *Knowledge-Based Systems*, *75*, 104–123. doi:10.1016/j.knosys.2014.11.023.

Silvertown, J. (2009). A new dawn for citizen science. *Trends in Ecology & Evolution*, *24*, 467 – 471. doi:10.1016/j.tree.2009.03.017.

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* (pp. 1225–1236). ACM.

Strigaro, D., Cannata, M., & Antonovic, M. (2019). Boosting a weather monitoring system in low income economies using open and non-conventional systems: Data quality analysis. *Sensors*, *19*. doi:10.3390/s19051185.

Sumsal, F., Brester, S. G., Szépe, V., & Halchenko, Y. (2005). Fail2ban. http://www.fail2ban.org/. Accessed 12-December-2016.

Sun, C., Ippel, L., Wouters, B., van Soest, J., Malic, A., Adekunle, O., Berg, B. v. d., Puts, M., Mussmann, O., Koster, A. et al. (2018). Analyzing partitioned FAIR health data responsibly. https://arxiv.org/pdf/1812.00991.pdf. Accessed 26-March-2019.

Swain, N. R., Christensen, S. D., Snow, A. D., Dolder, H., Espinoza-Dávalos, 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. *Environmental Modelling & Software*, *85*, 11 – 26. doi:10.1016/j.envsoft.2016.08.003.

Tanenbaum, J., Williams, A., Desjardins, A., & Tanenbaum, K. (2013). Democratizing technology: pleasure, utility and expressiveness in DIY and maker practice. In *Proceedings SIGCHI Conf Human Factors in Computing Systems* (p. 2603–2612). ACM.

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)*. Asilomar, California, USA.: Online Proceedings.

UptimeRobot (2016). Uptimerobot. `https://uptimerobot.com`. Accessed 12-December-2016.

Upton, E., & Halfacree, G. (2014). *Raspberry Pi user guide*. Hoboken, NJ, USA: John Wiley & Sons.

Van Rossum, G., & Drake, F. L. (2003). *Python language reference manual*. Network Theory United Kingdom.

Villa, F., Athanasiadis, I. N., & Rizzoli, A. E. (2009). Modelling with knowledge: A review of emerging semantic approaches to environmental modelling. *Environmental Modelling & Software*, *24*, 577–587. doi:10.1016/j.envsoft.2008.09.009.

Villa, F., Balbi, S., Athanasiadis, I., & Caracciolo, C. (2017). Semantics for interoperability of distributed data and models: Foundations for better-connected information. *F1000Research*, *6*. doi:10.12688/f1000research.11638.1. [version 1; referees: 2 approved with reservations].

Volker Andres, M. U., Simon Jirka (2014). *OGC Sensor Observation Service 2.0 Hydrology Profile*. Best Practice Paper 14-004r1 Open Geospatial Consortium.

de Vos, M. (2017). *Interpreting natural language spreadsheets*. phdthesis Vrije Universiteit. URL: `https://research.vu.nl/ws/portalfiles/portal/42791056/complete+dissertation.pdf`.

de Vos, M., Wielemaker, J., Rijgersberg, H., Schreiber, G., Wielinga, B., & Top, J. (2017). Combining information on structure and content to automatically annotate natural science spreadsheets. *International Journal of Human-Computer Studies*, *103*, 63 – 76. URL: `http://www.sciencedirect.com/science/article/pii/S1071581917300204`. doi:10.1016/j.ijhcs.2017.02.006.

Vretanos, P. P. A. (2014). *OGC Web Feature Service 2.0*. Interface Standard 09-025r2 Open Geospatial Consortium.

Vujović, V., & Maksimović, M. (2015). Raspberry Pi as a Sensor Web node for home automation. *Computers & Electrical Engineering*, *44*, 153–171.

Wikipedia contributors (2018). 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 7-August-2018.

Wikipedia contributors (2019). Raspberry Pi. `https://en.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=887687839`. Accessed 17-March-2019.

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. *Scientific data*, *3*. doi: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. *Agricultural Finance Review*, *76*, 15–26. doi:10.1108/afr-03-2016-0018.

Yazar, D., & Dunkels, A. (2009). Efficient application integration in IP-based Sensor Networks. In *Proc. 1st ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings* BuildSys '09 (pp. 43–48). New York, NY, USA: ACM. doi:10.1145/1810279.1810289.

Yu, L., & Liu, Y. (2015). Using Linked Data in a heterogeneous Sensor Web: challenges, experiments and lessons learned. *International Journal of Digital Earth*, *8*, 17–37.

Ziébelin, D., Hobus, K., Genoud, P., & Bouveret, S. (2017). Heterogeneous data integration using Web of Data technologies. In D. Brosset, C. Claramunt, X. Li, & T. Wang (Eds.), *Web and Wireless Geographical Information Systems* (pp. 35–47). Springer International Publishing.

Summary

This thesis investigates the state of the environmental data lifecycle in the Internet of Things era. We focus on two IoT stressors: a) the constraint resources ecosystem and b) the syntactic and semantic heterogeneity, and investigate their impact on environmental timeseries storage, dissemination, acquisition and integration. We argue that heterogeneity along with the low-capabilities of IoT devices, render past best-practices on environmental timeseries lifecycle not directly applicable. This thesis addresses the following research questions:

a. Can environmental timeseries lifecycle be facilitated by IoT prototyping devices?
b. Are environmental data dissemination protocols IoT-ready?
c. How can e-scientists acquire, integrate and transform environmental timeseries datasets in the heterogeneous IoT ecosystem?

In the light of the IoT resource-constrained ecosystem, we investigate whether a) the IoT prototyping devices can facilitate the environmental timeseries lifecycle, and b) environmental data dissemination protocols are IoT-ready. Chapter 2 presents our research to support resilient data storage on IoT prototyping devices. We focus on Raspberry Pi as an IoT prototyping device and explore its capabilities for resilient data storage, interoperable data dissemination through established standards and performance under concurrent requests from external clients. Chapter 3 presents our efforts to transform an established, environmental data dissemination protocol to be IoT compatible. We focus on OGC Sensor Observation Service (SOS) and argue that it was not designed to operate efficiently in the IoT enabling ecosystem. We designed and implemented a backwards-compatible extension which renders OGC SOS disruption-tolerant and supports for resource economizing.

In the light of data heterogeneity which is amplified in the IoT era, we explore new methodologies to support e-scientists towards acquiring, integrating and transforming environmental timeseries datasets. We argue that current approaches to facilitate aforementioned environmental data lifecycle processes have certain limitations. This is why, on top of the legacy environmental datasets, come new IoT-produced datasets which are increasingly used in environmental campaigns. These, are not always properly annotated and/or they report their data in custom formats, which render contemporary data acquisition and integration approaches not directly applicable. Chapter 4 reviews these approaches and proposes a declarative one to support e-scientists towards universal acquisition and integration of syntactically heterogeneous timeseries datasets. Our declarative approach is founded on templates, which are abstract descriptions of a dataset's syntax using programming language-agnostic semantics. We argue that templates offer a compromise between generality and simplicity, as e-scientists with different computer literacy profiles can develop them. We demonstrate

the syntactic interoperability capabilities of our approach with several case studies spanning across different environmental domains (i.e. meteorology, agriculture, urban air quality and hydrology). Chapter 5 extends this declarative approach with a reasoner to support semantic operations. We focus on one semantic heterogeneity challenge, that is the different units of measurement according to which observables are reported. Using user-defined semantic annotations, the reasoner determines the compatibility among datasets that are a) formatted with different syntaxes, b) annotated with custom semantics and c) reported with different units of measurement. We demonstrate the semantic interoperability capabilities of our approach in a case study where we transform meteorological syntactically and semantically heterogeneous input files of four agricultural models, performing (when applicable) the on-the-fly units of measurement transformation.

Chapter 6 concludes this thesis and summarizes its main contributions, which are regarded with:

a. providing with insights about the limits of contemporary IoT gateways and their performance as active participants in the environmental data lifecycle (Chapter 2),
b. developing an IoT-ready, backwards compatible extension for the OGC SOS to support interoperable data dissemination on-site (Chapter 3),
c. designing and implementing a declarative approach which facilitates the acquisition, transformation and integration of syntactically (Chapter 4) and semantically (Chapter 5) heterogeneous environmental datasets.

Acknowledgements

The past five years have been a rollercoaster ride. Everything started in Xanthi and finished at Wageningen. They say that a PhD is a lone endeavour, but I was lucky enough to have a selected few who supported me throughout these years.

First and foremost, I would like to express my gratitude to Dr. Ioannis Athanasiadis. Ioannis and I started collaborating in 2014, when I began my bachelor thesis in Xanthi. Since then, a lot has changed, but our fruitful collaboration still carries on. Thank you Ioannis for the opportunities you offered me, our discussions, your comments on our manuscripts, to name a few occasions. During the past five years and thanks to your mentorship, not only did I develop new skills but I also became a better researcher. I would also like to thank my promotor Dr. Bedir Tekinerdogan. Thank you so much Bedir for your guidance, comments and for always making me feel like home, when I visited Wageningen for research.

Having worked in different projects throughout the years, I was lucky enough to collaborate with exceptional people. Thank you so much Pierluigi, Tommy and Gianni for our collaboration in the context of MODEXTREME. I have the best memories from our inspiring project meetings. I am also deeply honored to have been considered a co-author in our paper. Marcelo, Simone and Gaetano it is amazing to have met and worked with you. I highly enjoyed working with you in the context of MODEXTREME and during my (short) involvement with the BioMA platform. The International Spring University was my first adventure in this academic journey. Thank you Ferdinando, Stefano, Brian and Ken for the unforgettable one-week training on Ecosystem Services Modeling.

In September 2018 I joined the ING IT Class. During this amazing journey, I met friends and worked with great colleagues. I would like to thank all my friends and colleagues at ING and BMG. Marc, Luuk, Sil, Lukas, Jurriaan, Hans, Vincent, Nico, Robert, Patrick, Yoeri, Chris, Theuns, Andrea, Nedim, Daniel, Nikita, Dmitrii, Bo, Miel, Olle and Salad: It was a pleasure meeting and working with you.

Without the help of friends and colleagues this journey would have been very difficult. Thank you Stavros for your unconditional help and our geek discussions. I met you in Xanthi by chance and you are by-far the coolest geek I know! Sotiris and Simos thank you for our small talks and deep discussions during our coffee breaks (and drinks) in Xanthi. Avi, thank you so much for our collaboration during my "transition" period in Xanthi. Thank you Evangelia, for our joint efforts in CEDAR and EDAM. I hope you will be finalizing your own thesis soon. I keep saying that I don't have friends, but only acquaintances: Michalis, Kostis, Kosmas, Apostolos and Kostakis, if I ever get friends, you will be my first options! Thank you so much for your support and acquaintanceship all these years.

The biggest merit of this journey is attributed to my family. You never stopped supporting me in whichever decision I ever made. You are always happier than I am for my successes and sadder than I am when something goes wrong. A "thank you" is not enough, thus naturally this thesis is dedicated to you. Ευχαριστώ πολύ για όλα (αδερφέ) Άγγελε, (μαμά) Βάνα, (γιαγιά) Βάσω, (μπαμπά) Γιώργο, (παππού) Λάκη και (θεία) Ντίνα. Αυτή η διατριβή είναι αφιερωμένη σε σας.

Paschalina thank you so much for being my partner in this rollercoaster journey. You have been a part of this since the very beginning giving me the strength to move forward. Without you, this whole journey would have been less fun and more difficult. I am grateful for your patience and constant motivation.

List of publications

Peer-reviewed journal publications

Samourkasidis, A., Papoutsoglou, E., & Athanasiadis, I. N. (2018). A template framework for environmental timeseries data acquisition. *Environmental Modelling & Software*, *117*, 237–249. `https://doi.org/10.1016/j.envsoft.2018.10.009`

Klein, T., **Samourkasidis, A.**, Athanasiadis, I. N., Bellocchi, G., & Calanca, P. (2017). WebXTREME: R-based web tool for calculating agroclimatic indices of extreme events. *Computers and Electronics in Agriculture*, *136*, 111–116. `https://doi.org/10.1016/j.compag.2017.03.002`

Samourkasidis, A., & Athanasiadis, I. N. (2017a). A miniature data repository on a Raspberry Pi. *Electronics*, *6*(1). `https://doi.org/10.3390/electronics6010001`

Peer-reviewed book chapter

Samourkasidis, A., & Athanasiadis, I. N. (2017b). A Sensor Observation Service Extension for Internet of Things. In Podnar Žarko I., Broering A., Soursos S., & Serrano M. (Eds.), *2nd Intl. Workshop on Interoperability and Open-Source Solutions for the Internet of Things (InterOSS-IoT)* (Vol. 10218, pp. 56–71). Springer, Cham; Springer International Publishing. `https://doi.org/10.1007/978-3-319-56877-5_4`

Other scientific publications

Samourkasidis, A., & Athanasiadis, I. N. (2018b). Towards a semantic approach for environmental timeseries data re-usability. In *Scientific Symposium FAIR Data Sciences for Green Life Sciences.*

Samourkasidis, A., & Athanasiadis, I. N. (2018). Environmental timeseries lifecycle in the IoT era. In M. Arabi, O. David, J. Carlson, & D. P. Ames (Eds.), *In Proceedings of the 9th Intl. Congress on Environmental Modelling and Software (iEMSs).*

Samourkasidis, A., & Athanasiadis, I. N. (2016). OGC Sensor Observation Service: Past, present, Internet of the Things. In S. Sauvage, J. Sánchez-Pérez, & A. Rizzoli (Eds.), *In*

Proceedings of the 8th Intl Congress on Environmental Modelling and Software (iEMSs) (Vol. 1, p. 229).

Klein, T., **Samourkasidis, A.**, Athanasiadis, I. N., Bellocchi, G., & Calanca, P. (2016). webXTREME: A simple web tool for calculating agroclimatic indicators of extreme events. In S. Sauvage, J. Sánchez-Pérez, & A. Rizzoli (Eds.), *In Proceedings of the 8th Intl Congress on Environmental Modelling and Software (iEMSs)* (Vol. 2, p. 471).

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 V. K. Johannsen, S. Jensen, V. Wohlgemuth, C. Preist, & E. Eriksson (Eds.), *Adjunct proc. 29th EnviroInfo and 3rd ICT4S conference* (pp. 65–74). University of Copenhagen.

Samourkasidis, A., & Athanasiadis, I. N. (2014). Towards a low-cost, full-service air quality data archival system. In *In Proceedings of the 7th Intl. Congress on Environmental Modelling and Software, International Environmental Modelling and Software Society (iEMSs).*

Siafarikas, D., **Samourkasidis, A.**, & Arampatzis, A. (2014). A cost-benefit analysis of indexing Big Data with map-reduce. In *In Proceedings of the 7th Electrical and Computer Engineering Student Conference (ECESCON)* (pp. 165–170).

About the author

Argyrios Samourkasidis was born in Komotini, Greece on August 12th, 1991. His research interests are in the areas of Internet of Things, knowledge representation, data interoperability, web services, automation and cyber security. Argyrios' latest publications can be found at Google Scholar (https://scholar.google.com/citations?user=xVL9r80AAAAJ&hl=en).

Argyris holds an MSc in Electrical and Computer Engineering (5-year engineering curriculum) from Democritus University of Thrace (top 10% of his class). During his thesis which revolves around an "autonomous system for fusion, preprocessing and dissemination of air-quality environmental data on the semantic web", Argyris discovered his interest in environmental data and Internet of Things.

In 2014 Argyris started his PhD research at the Democritus University of Thrace. During this period, Argyris worked as a researcher in ALPINE, a Greek General Secretariat of Research and Technology (GSRT) project, and MODEXTREME, a European Commission (FP7) funded project. In the context of ALPINE, he was occupied with the acquisition, processing and annotation of hydrological data stream from sensor feeds, in order to be suitable for semantic modelling. In the context of MODEXTREME, Argyris maintained the project's website and assisted in the design and dissemination of newsletters and online questionnaires, as well as getting insights from them. During his occupation for the two projects, Argyris contributed in writing ten deliverables in total.

In 2016 Argyris joined the Information Technology Group in Wageningen University as an external PhD student. During his PhD, Argyris has contributed three journal, one book chapter and four conference, peer-reviewed publications. He attended four international conferences and one international symposium delivering eight presentations. During the iEMSs 2016 conference Argyris co-organized the "From Environmental Information Systems to Big Data" workshop.

In September 2018 Argyris joined ING as an IT Class member. During his assignments there as a DevOps engineer he gained experience in cyber security, cryptography and site reliability engineering. He is a senior Python developer with exposure to Pandas, Flask, Ansible, Java and Spring. Argyris holds a Junior Penetration Tester, COBIT 5 Foundation and Ecosystem Services Modeling certificates issued by eLearnSecurity, ISACA and BC3, respectively.

Colophon
The research leading to this thesis was partially funded by the European Community's Seventh Framework Programme (FP7/2007-2013) within the project MODEXTREME under grant agreement n° 613817 and Greek General Secretariat for Research and Technology within the project ALPINE under grant agreement n° 11SYN_6_411.

Cover Design by: Remco Wetzels

Printed by: Proefschriftmaken.nl

Propositions

1. Without standards Internet of Things devices produce legacy datasets which require customized curation. (this thesis)
2. Internet of Things devices contribute towards democratizing e-science and lowering its entry barriers. (this thesis)
3. Citizen Science and Open Access are two sides of the same coin.
4. The Dunning-Kruger effect curve depicts a typical PhD study.
5. When business is involved in scientific research, journal impact factors and community sizes are affected.
6. People looking at their screens during conference presentations or even conversations is a sign of our times.

Propositions belonging to the thesis, entitled

Environmental timeseries lifecycle in the Internet of Things era
Lowering e-science barriers

Argyrios Samourkasidis
Wageningen, 14 October 2019