TECHNISCHE UNIVERSITÄT WIEN

DISSERTATION

Heterogeneous building related data streams for performance assessment applications

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der Technischen Wissenschaften

unter der Leitung von

Univ. Prof. Dipl.-Ing. Dr. techn. Ardeshir Mahdavi

E259-3 Abteilung für Bauphysik und Bauökologie

Institut für Architekturwissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Architektur und Raumplanung

von

Dipl.-Ing. Dawid Wolosiuk

Matrikelnummer: 01127262

Wien, am 29 Juli 2021

Acknowledgements

First and foremost, I would like to thank Professor Mahdavi for his guidance, support and encouragement both in this research project in particular and during our overall collaboration at the TU Wien.

I would also like to acknowledge my current and former colleagues from the Department of Building Physics and Building Ecology for their support both on professional as well as personal level.

Finally, my heartfelt thanks go to my wife Anna and my wonderful family for their love and always being there for me. Without their support I could not have reached where I am now.

Abstract

Ontologies are developed and deployed to enhance knowledge and data exchange in a specific field or domain. For instance, building industry benefits from well-structured ontologies such as Industry Foundation Classes (IFC) or green building XML that drive Building Information Modeling software (BIM). In this context, efforts in building performance specification and assessment can also benefit from well-structured ontologies and data schemas. Toward this end, a recently introduced building performance data ontology (complemented with building performance data schema) attempts to identify, categorize, and capture the complexities of building related performance data and its attributes.

There are many different types of building-related data, each requiring a specific approach toward creation and implementation of fitting ontologies. In case of building performance data streams originating, for instance, from sensors or simulations, these are often structurally syntactically or semantically heterogeneous and could benefit from an integration process. Such process of ontological data integration involves i) preprocessing, ii) categorical identification, iii) supplementation of the relevant attributes (either via templates, or data scraping), and iv) encoding in a proper file format. Only then is such ontologized data ready be used in various downstream applications. This effort discusses the concept of ontologies and its current role in built environment domain. It describes an ontologization process as applied to a large real-world building monitoring dataset. Specifically, monitored environmental data are first processed in terms of fidelity and quality to be subsequently ontologized and delivered to a number of building performance assessment applications.

Kurzfassung

Ontologien werden entwickelt und eingesetzt, um den Wissens- und Datenaustausch in einem bestimmten Bereich oder einer bestimmten Domäne zu verbessern. Die Bauindustrie profitiert beispielsweise von gut strukturierten Ontologien wie Industry Foundation Classes (IFC) oder Green Building XML, welche die Building Information Modeling Software (BIM) vorantreiben. In diesem Zusammenhang können auch Bemühungen zur Erstellung von Leistungsspezifikationen und -bewertungen von gut strukturierten Ontologien und Datenschemata profitieren. Zu diesem Zweck versucht eine kürzlich eingeführte Ontologie von Gebäudeleistungsdaten (ergänzt mit einem Schema für Gebäudeleistungsdaten), die Komplexität von gebäudebezogenen Leistungsdaten und ihren Attributen zu identifizieren, zu kategorisieren und zu erfassen.

Es gibt viele verschiedene Arten von gebäudebezogenen Daten, die jeweils einen spezifischen Ansatz für die Erstellung und Implementierung passender Ontologien erfordern. Bei Gebäudeleistungsdatenströmen, die beispielsweise aus Sensoren oder Simulationen stammen, sind diese oft strukturell syntaktisch oder semantisch heterogen und könnten von einem Integrationsprozess profitieren. Ein solcher Prozess der ontologischen Datenintegration umfasst i) Vorverarbeitung, ii) kategoriale Identifizierung, iii) Ergänzung der relevanten Attribute (entweder über Vorlagen oder Daten-Scraping) und iv) Codieren in ein geeignetes Dateiformat. Erst dann können solche ontologisierten Daten in verschiedenen nachgelagerten Anwendungen verwendet werden. Dieser Versuch diskutiert das Konzept von Ontologien und seine aktuelle Rolle im Bereich der gebauten Umgebung. Es beschreibt einen Ontologisierungsprozess, welcher für auf einen großen realen Gebäudeüberwachungsdatensatz angewendet wird. Konkret werden überwachte Umweltdaten zuerst in Bezug auf Genauigkeit und Qualität verarbeitet, um anschließend ontologisiert und an eine Reihe von Anwendungen zur Gebäudeleistungsbewertung geliefert zu werden.

Contents

CHAPTER 1

Introduction

1.1. Motivation

The rapid advances in computer and information technology over the past few decades have resulted in unprecedented developments in science, technology, business and economy. One of the most enabling elements as well as a by-product of these advancements is data, that has since grown exponentially in volume and equally in importance. Naturally with growing data volume, the variability of available data type, format and storage instances has also increased considerably. This poses a problem for data accessibility or reusability in the context of the heterogeneity of the data sources. Such accessibility issues even occur within a specific interest group working on the same subject matter (Ushold and King 1996). This is due to the fact that the actors involved in use and generation of data very often have different needs, backgrounds, contexts, viewpoints and assumptions. Typically, such circumstances result in data structures that are often tied to a particular organization, system, or even a single application instance, and the prospect of reusability is very limited - a phenomenon known as a *data silo*. It became apparent that making data reusable and more accessible can substantially support management, operation, research or collaboration within a field of interest and beyond.

The importance of data accessibility was first recognized and addressed in research fields where computer and information technology were the most present, such as Artificial Intelligence, Software Engineering and Database Systems communities (Sánchez et al. 2007). As a result, data models (also called schemas) and ontologies were introduced. The

primary role of these concepts is to add meaning, structure and relationships to the data, thus making it more readable (accessible) to both users and machines.

With the information and communication technologies progressively becoming integral part of other disciplines, domains, and fields, a similar need arose for a systematic representation of knowledge and information, and better data management solutions within these. For example, if we look at the domains and fields associated with the built environment, the actors involved in the building process work on the same matter, but are involved in different aspects and phases of this process, such as: E.g. design, analysis, construction, maintenance or operation. Naturally, each of these phases nowadays is enabled and driven by information technology and involves multiple stakeholders. As a result, the related information space also extends over multiple domains and scales. This brings us back to the problem of limited accessibility and usability due to multiplicity (hence heterogeneity) of data streams. These are determined by hardware specifications, software standards and formats, or system specifications.

The Architecture, Engineering, and Construction (AEC) community already benefits from common data models that are driving the Building Information Modelling (BIM) and integrate efforts of multiple stakeholders. Such integration has been suggested to facilitate better communication between parties involved and to improve the overall efficiency of the building design, construction, and operation processes. However, the main focus of BIM related schemas, lies in the representation of primarily static building attributes, including geometry and semantic information on building components and systems.

Buildings are increasingly equipped with sophisticated monitoring infrastructures, recently boosted by technological advancements in wireless sensor networks and low-power microcontrollers fields. These collect a large volume of multiple layers of dynamic data on the states and events related to building systems' performance, indoor and outdoor environmental conditions, or occupants' location, movement, and control-oriented actions (Wolosiuk and Mahdavi 2020; Mahdavi and Wolosiuk 2021). In order to effectively support the evidence-based design, assessment and operation that rely on (and benefit from) such building related dynamic data streams, relevant data schemas and ontologies must be developed and implemented.

Building performance is important throughout the entire building's life cycle, form the design phase to construction, maintenance and ultimately disposal (De Wilde 2018). Majority of building related dynamic data sources (e.g. obtained from monitoring systems or simulation software) can be considered as Building Performance Data (BPD). These occur in the form of performance indicators or as performance variables/measures that compose a complex indicator. Building performance assessment procedures typically make use of a large number of BPD, involving multiple domains, aspects, and degrees of resolution. The use cases of BPD are diverse. These include, intelligent building operation, smart grid applications, compliance demonstration with building code requirements,

specification of building attributes in certificate-type documents, as well as comparison and ranking of building design alternatives. However, despite the extensive use of BPD (both performance measures and indicators), there have been very few attempts to compose an explicit BPD ontology (e.g. Mahdavi et al. 2005, Corry et al. 2015). A versatile BPD ontology can add to the clarity of building performance requirements specifications, advance the understanding of building performance principles, and provide a solid foundation for the development of wide range of relevant applications such as all-purpose data visualization engines or interfaces to variety of performance simulation tools (Mahdavi and Wolosiuk 2019a).

The present research effort addresses the paucity in comprehensive representation of the wide scope of building related performance data originating from heterogeneous data sources. In this context, the Building Performance Data ontology was proposed in an attempt to organize knowledge in this domain and capture the complexities of building related performance data and its attributes in a robust data schema.

1.2. Background

The work included in this dissertation is a combination of knowledge and methods taken from the computer, information and architectural sciences. The latter provides the subject matter – that is the domain knowledge concerning building performance data and related issues. The former two provide the foundation for presented research in form of ontologies and schemas, as well as means for implementation and application illustration.

The modern concept of ontology that is discussed in this work originates from the Ontology – the branch of philosophy dealing with nature and structure of reality (Guarino et al. 2009). There is a vast literature on theoretical foundations of ontologies and how the philosophical concept got adapted in field of Computer Science, for example see Sanchez et al. (2007), Guarino et al. (2009), Gruber (2009) or Gómez-Pérez et al. (2010).

Probably the most commonly cited early definition of computational ontology is that by Gruber (1993):

> "… A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are presumed to exist in some area of interest and the relationships that hold them (Genesereth and Nilsson 1987). A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly. **An ontology is an explicit specification of a conceptualization**. …"

Initially the computational ontologies and data schemas have been developed in response to growing need for knowledge representation frameworks to support evolution in

computer sciences – in particular artificial intelligence, software engineering and database systems. Fundamentally, ontologies are expected to help organize and structure information to enable a common and shared understanding of a particular domain. Such an understanding leads to better communication, enables interoperability and re-usability (Uschold and Gruniger 1996).

Ontologies facilitate the integration of data, information and knowledge by formalizing the vocabulary, specifying the hierarchy of relevant concepts (or classes), supplying components with relevant attributes and defining relationships between them. Currently arguably the most prominent exemplification of ontology use is the Semantic Web (Berners-Lee 2001). It is the extension of existing World Wide Web that is supposed to give information a well-defined meaning by providing additional layer of machine understandable data. Ontologies are the backbone of the Semantic Web (Taniar and Rahayu 2006, Domingue et al. 2011). They facilitate automation and interoperability of web enabled applications and systems in areas such as search engines, social networks, digital recourses management or e-commerce.

There are several fields outside computer science domain where ontologies already play important role and are widely deployed, for example knowledge management or biomedical domains. In the field of knowledge management, they serve as the basis for the collection, integration and organization of information; they support the search, retrieval, personalization and visualization of knowledge (Abecker and van Elst 2009, Davies et al. 2002). Ontology-driven knowledge management has become a crucial factor for running an efficient and successful organization (Stabb et al. 2001).

The biomedical domain is known for the use of many well established ontologies. Due to a growing volume of data and a growing complexity as well as overlapping concepts in this domain, it was necessary to establish robust ontologies in order to drive progress in related areas. These ontologies help in unifying diverse datasets, creation of knowledge bases or establishing controlled vocabularies that enable interoperability, data exchange, knowledge retrieval and interpretation, or hypothesis evaluation (Rubin et al. 2008). Some examples of ontologies in biomedical domain are, i) the Gene Ontology (Ashburner et al 2000) for describing biological processes, molecular functions and cellular components of gene products, ii) The NCI Thesaurus (Hartel et al., 2005) that provides a controlled terminology that enables researchers to integrate, retrieve, and relate diverse data collected in cancer research or iii) The Foundational Model of Anatomy (Rosse and Mejino 2007) for the symbolic representation of the phenotypic structure of the human body. These and many other biomedical ontologies support insightful analyses and scientific discovery in this complex domain.

Accordingly, the need for data integration and interoperability has also been recognized in other areas, domains and communities in order to enable a feasible exchange of data and knowledge and benefit from all the associated advantages. This includes many fields and

areas related to buildings and built environment. There have been many attempts to establish ontologies and schemas pertaining to this domain. Probably the most established ontological data schemas are those primarily focused on "static" building data (that related to construction efforts), such as the Industry Foundation Classes (IFC) and the green building Extensible Markup Language (gbXML). Many other building related ontologies often address the need for knowledge organization in more specific areas of interest that potentially involve "dynamic" building data. These ontologies differ in scope and are often a response to paucities (both informational and functional) in the aforementioned primarily "static" schemas. Some of these schemas, especially the ones defined in RDF (Resource Description Framework) data model (Lassila and Swick 1999), adopt existing elements or semantics from other established schemas (including IFC and gbXML) to allow for some interoperability, standardization or re-use support.

Recently published review paper by Pritoni et al. (2021) reviews 40 schemas and ontologies for building energy applications. The authors categorized schemas according to phase of the building lifecycle. Namely, *Design and/or energy modeling* (e.g. IFC, gbXML), and building *Operations* divided into 5 application groups: i) Sensor networks, Internet of Things, and smart homes (e.g. DogOnt, SAREF), ii) Commercial building automation and monitoring (e.g. Project Haystack, Brick Schema, BOT), iii) Grid-interactive efficient building (GEB) applications (e.g. RESPOND (Esnaola-Gonzalez and Díez, 2020)), iv) Occupants and behavior (e.g. DNAS Framework (obXML)), and v) Asset management and audits (e.g. BuildingSync). This section presented the spectrum of relevant building domain ontologies (as demonstrated above).

The following section presents examples of relevant ontologies and schemas in most of the above-mentioned categories. Specifically: i)IFC for complex building modelling; ii)gbXML for energy modeling; iii) SAREF for smart home devices; iv) Project-Haystack schema for data related to smart devices in buildings; v) Brick schema to represent building data at large; vi) The Building Topology Ontology (BOT) for specification of relationships between components and zones of a building; vii) DNAS/obXML to capture occupant behavior in buildings; viii) BuildingSync schema for data related to energy audit.

Industry Foundation Classes

The **IFC** "are an open international standard for Building Information Model (BIM) data that are exchanged and shared among software applications used by the various participants in the construction or facility management industry sector" (IFC ISO 2018). It specifies data schema and file format used for creating digital description of the built environment. It has been developed since 1995 by buildingSMART International (bSI) (formerly - International Alliance for Interoperability) gradually building recognition and position in the AEC community to become leading data exchange format for the AEC industry that enables the BIM. Currently over 800 industry members such as organizations, companies and institutes that participate in development and promotion of bSI standards (Boreman

et al. 2018). Since 2013 it is registered with ISO as ISO 16739 which enhanced format's credibility even more. As a result, nowadays many countries require adoption of BIM on publicly procured projects.

The IFC format is supported by multiple BIM related software for modelling, managing or simulating the built environment (341 applications listed as of May 2021; buildingSMART, 2021a). Following the schema's specification, the real-world objects and actions such as building construction elements, building systems' elements, construction schedules or cost estimates are abstracted into entities and given required and optional attributes, properties and relationships to other entities. For encoding model into a text representation, it utilizes the EXPRESS data modeling language (Schenck and Wilson 1993). In parallel, the schema offers more accessible ifcXML format specification, but is not commonly used as a data exchange format due to much larger size of already "heavy" default file format, rendering it unpractical for complex building models.

The current official version of IFC data schema (version 4.0.2.1; buildingSMART 2021b) includes definitions of over a 1000 entities, enumerations and measure types put into the *resource, core*, *interoperability* and *domain* specific schema architecture layers (see Figure 1 for the schema architecture). Resource layer — contains resource definitions to describe higher level entities, core layer — contains entity definitions, provides the basic structure and relationships, interoperability layer — contains entity definitions for inter-domain information exchange, domain layer — contains entity definitions pertaining to a specific domain for intra-domain information exchange (buildingSMART, 2021b).

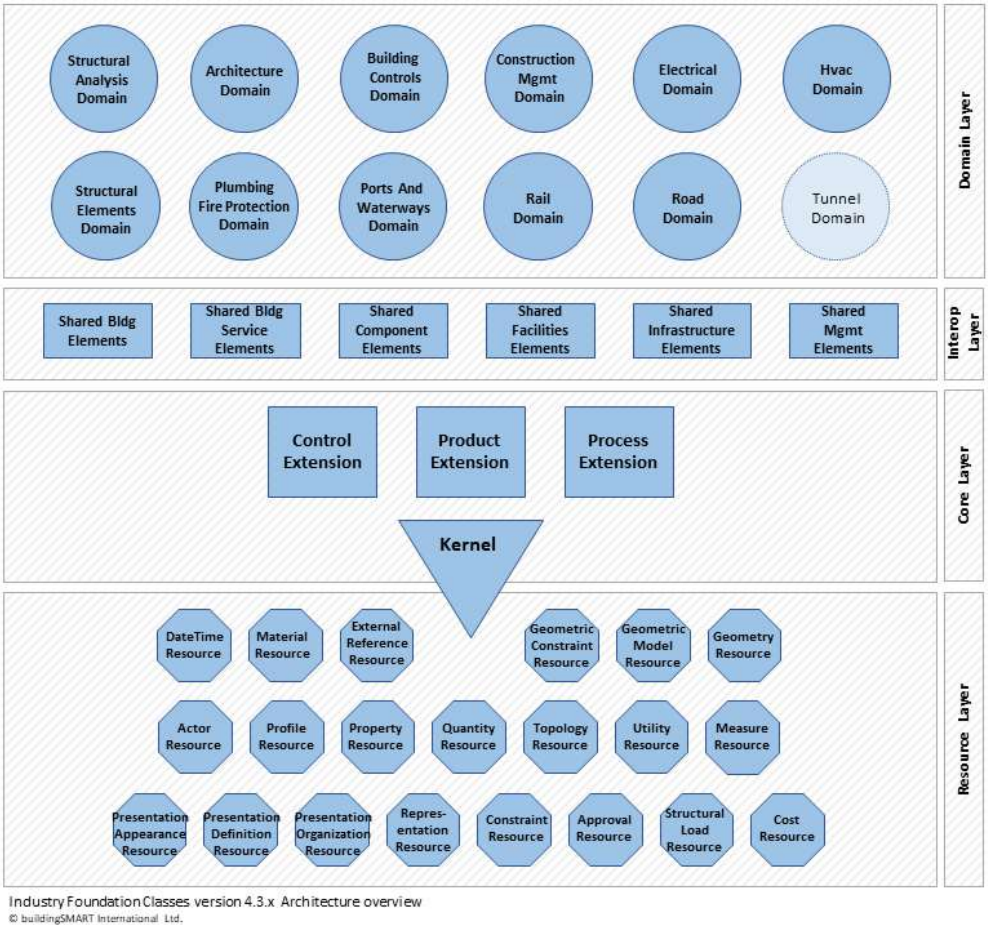

Figure 1 Industry Foundation Classes architecture overview. (source: buildingSmart 2021b)

Green Building XML

In contrast to IFC which is primarily oriented towards construction and management process, the gbXML is a data schema primarily oriented towards support of building performance analysis software. It was developed to facilitate the design of more energy efficient buildings and high-quality indoor environments.

It is an open-source schema that has been developed by a non-profit organization - the Open Green Building XML Schema, Inc. Its maintenance and development are supported by public institutions (e.g. U.S.DOE, NREL) as well as leading CAD-based BIM and performance analysis software companies (e.g. Autodesk, Bentley, IES) (ref to gbxml.org). It has been developed since 1999 and is now considered the industry standard schema for sharing information between BIM software and analysis tools. It is focused on storage of building related information potentially needed by simulation software to enable different types of building analysis such as energy use, CFD, heating and cooling load, lighting, solar, shading, HVAC sizing etc. The open source aspect and use of simple and robust XML

(both human- and machine- readable format) for information encoding makes implementation of gbXML in analysis tools relatively easy, hence it is currently supported by over 50 different applications and tools.

The schema defines over 500 different types of elements (similar to entities in the IDF format), attributes and enumerations to describe building model. These include among others: 3D and 2D geometry, construction elements and materials, space boundaries, internal and external equipment, lighting characteristics and control, occupancy or schedules. A serialized gbXML file contains elements defined by schema that are connected with other related elements and attributes in a hierarchical manner. In certain cases (such as boundary condition definition) the relationships to other elements are defined through specific attributes.

SAREF

The Smart Applications REFerence ontology (SAREF) (ETSI 2020) was created to enable the interoperability between various Internet of Things (IoT) devices by matching their existing assets (standards, protocols, data models etc.). Figure 2 shows an overview of the main classes and relationships in SAREF ontology. To accomplish the said interoperability, it describes IoT *devices* as objects (e.g. appliance, actuator, sensors, meter, HVAC) designed to perform certain tasks and to accomplish this task it performs certain *function* (e.g. actuating, sensing, metering, event) that have associated *commands* (e.g. switch on, switch off, toggle) and a *state* that it is in. Device can offer *service* that represents a way to communicate or control other devices in the network (EU Commission and TNO 2015). For example, an HVAC device can offer a temperature sensor to control its on/off function.

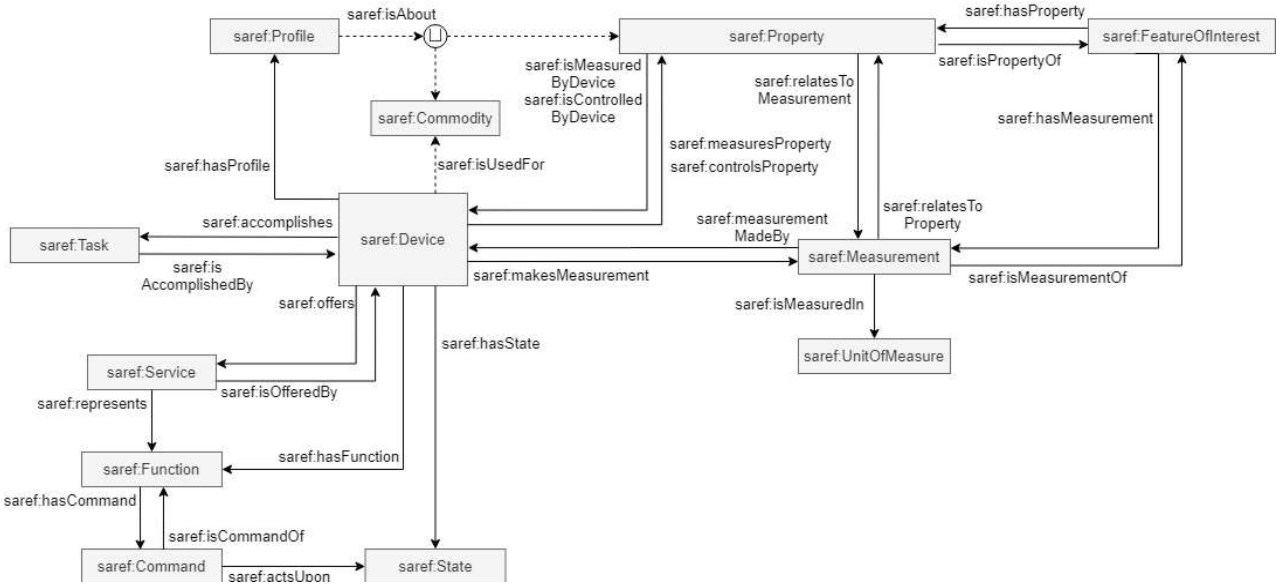

Figure 2 Overview of the SAREF ontology (source: ETSI 2020)

Project Haystack

Project Haystack (2021a) created a schema that addressed the need for a data model for smart devices and equipment systems present in built environment to make related data more meaningful or "self-describing". ProjectHaystack is an open source initiative backed by major companies such as Intel, Simens, or LaGrand. By offering toolkits, communication protocol of a ready server implementations on top of the schema it has become de-facto a standard for streamlining equipment data to building automation systems.

The schema defines a vocabulary of relevant "tags" (or metadata) of different type that are used to abstract real-world devices and equipment (such as electric meters, HVAC units, temperature sensors, pressure sensors or on/off switches) to entities. Different combinations of tags are used to characterize an entity depending on what they represent.

Principally an entity has a set of general identification attributes, a set of descriptive/ informational tags specific to the entity type and a set of reference tags specifying relationships between the entities. A resulting Project-Haystack building model is a combination of sensors' data point entities that are associated with equipment entities that are further associated with a location or site entities (see Figure 3 for an illustrative example) (Project Haystack 2021b).

Brick

Brick (Balaji et al 2018) is an open source project that was initiated in 2015 by academic community. Similar to the project Haystack – it is an attempt to create a metadata schema for smart buildings. The Brick schema captures physical, logical and virtual entities in buildings, contextualize and attributes them, group them in a class hierarchy, and define how these entities relate to one another. It is not intended to replace existing standards used in building management or automation systems but rather to make the existing data more exposed and available.

The brick is built on three main categorical classes used for grouping entities: i) *Point*, ii) *Equipment* and iii) *Location*. The *Point* class host different sources of data such as sensors, setpoints, status etc. The *Equipment* class represents devices and installations related to HVAC, fire safety, solar power, lighting, etc. The *Location* class include real and logical locations in the building, such as site, floors, rooms, zones etc. Figure 4 presents an illustrative example of entities and their relationships as defined by the Brick model. A resulting Brick model is a representation of assets relationships and data in a building (Brick Consortium 2021a, 2021b).

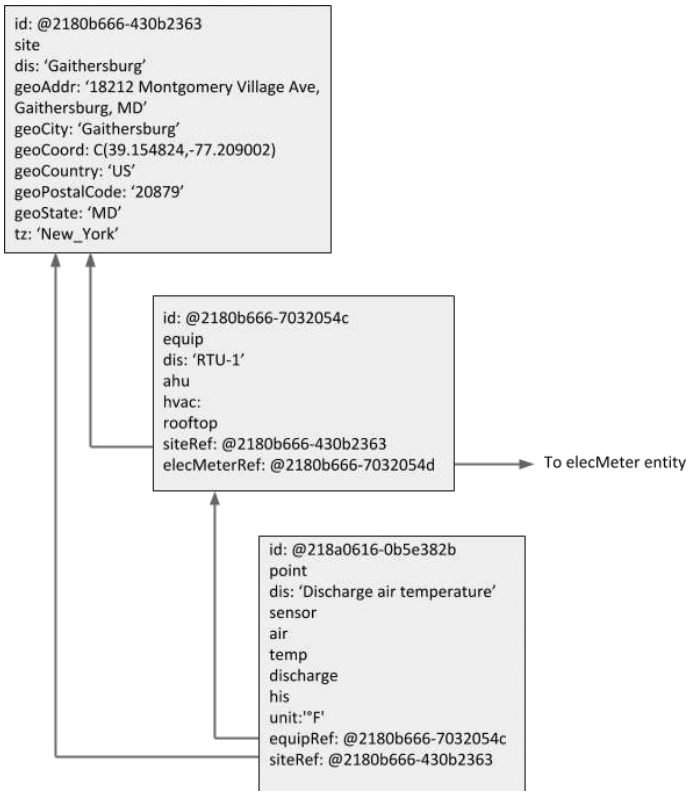

Figure 3 Diagram of ProjectHaystack entities and their relationships. Here, a temperature sensor (point) associated with a HVAC Unit (equipment) located in a building in Gaithersburg, USA (site). (source: Project Haystack, 2021b)

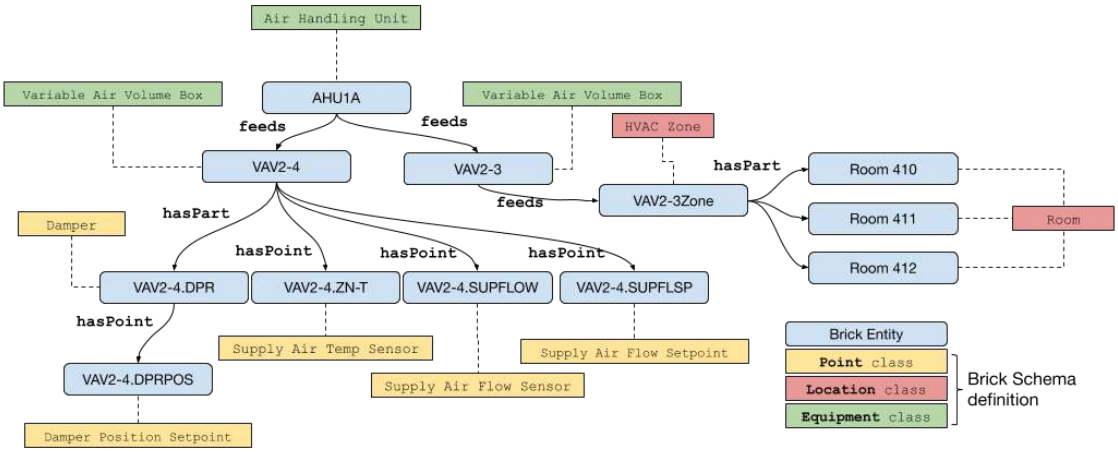

Figure 4 Illustrative example of entities and their relationships as defined by the Brick model. (source: Brick Consortium 2021b)

Building Topology Ontology

The Building Topology Ontology (BOT) (Rasmussen et al. 2017a, 2017b) was developed to provide "a high-level description of the topology of buildings including stories and spaces, the building elements they may contain, and the 3D mesh geometry of these spaces and elements" (Rasmussen et al., 2020).

Currently, the BOT is developed by the World Wide Web Consortium (W3C) Linked Building Data Community Group (W3C 2021). The group gathers BIM and Web of Data technology experts together to bring the Linked Data and Semantic Web Technologies (Domingue et al. 2011) to the AECOO (Architecture, Engineering, Construction, Owner and Operation) industry for greater integration of data and extended interoperability between building related and various web-enabled datasets from other communities beyond building domain (W3C 2021, Rasmussen et al. 2017a).

BOT ontology specifies four main classes to describe a building – *zones*, *elements*, *interfaces* and *3Dmodel*. Zones class defines parts of physical and virtual world in built context such as site, story, space, thermal zone. Elements class defines any physical object in a zone such as construction element, device, installation element, furniture etc. Interfaces defines common parts between elements and zones in the building. 3Dmodel class is used to link zones and elements to a 3D model (stored in a 3D file format).

As a lightweight ontology it is intended to be easily linked and used with other ontologies and function as a core element of an ontological building model. Such interoperability with other ontologies (e.g. those describing performance data, sensors, equipment, systems, management data, product information) is enabled by its implementation using Semantic Web technologies such as the RDF (Lassila and Swick 1999), or the Web Ontology Language OWL (Hitzler et al. 2009; Rasmussen et al. 2020).

DNAS/obXML

The DNAS (Drivers, Needs, Actions and Systems) (Hong et al. 2015a) ontology was developed as an attempt to capture energy-related occupant behavior in buildings. Occupant behavior is difficult to define and quantify and yet it can have major impact on energy usage.

The DNAS ontology is based on human-building interaction framework components described in Turner and Hong (2013). These include the *drivers* of behavior (external factors that provoke energy related occupant behavior), *needs* (requirements ensuring occupant's satisfaction in an environment), *actions* (activities or interactions with systems that occupant conducts to satisfy the needs) and *systems* (any sort of equipment mechanisms or other measures that an occupant interacts to control comfort or satisfaction in the environment) (Hong et al. 2015a).

The ontology was implemented in occupant behavior XML (**obXML**) schema (Hong et al. 2015b). The occupants' behavior is captured in the schema by elements pertaining to 3

main sub-classes of the *OcupantBehavior* root class. *Behaviors* class implements the aforementioned DNAS framework to capture complex patterns of occupants' behavior (see Figure 5 for an overview of the behaviors class), *Buildings* class provides special context for occupants' actions and finally *Occupants* class identifies and provide some relevant details on singular occupant within a building. Resulting standardized occupant behavior model built with DNAS/obXML schema enables, e.g., sharing information for better models' assessment or more accurate building simulation.

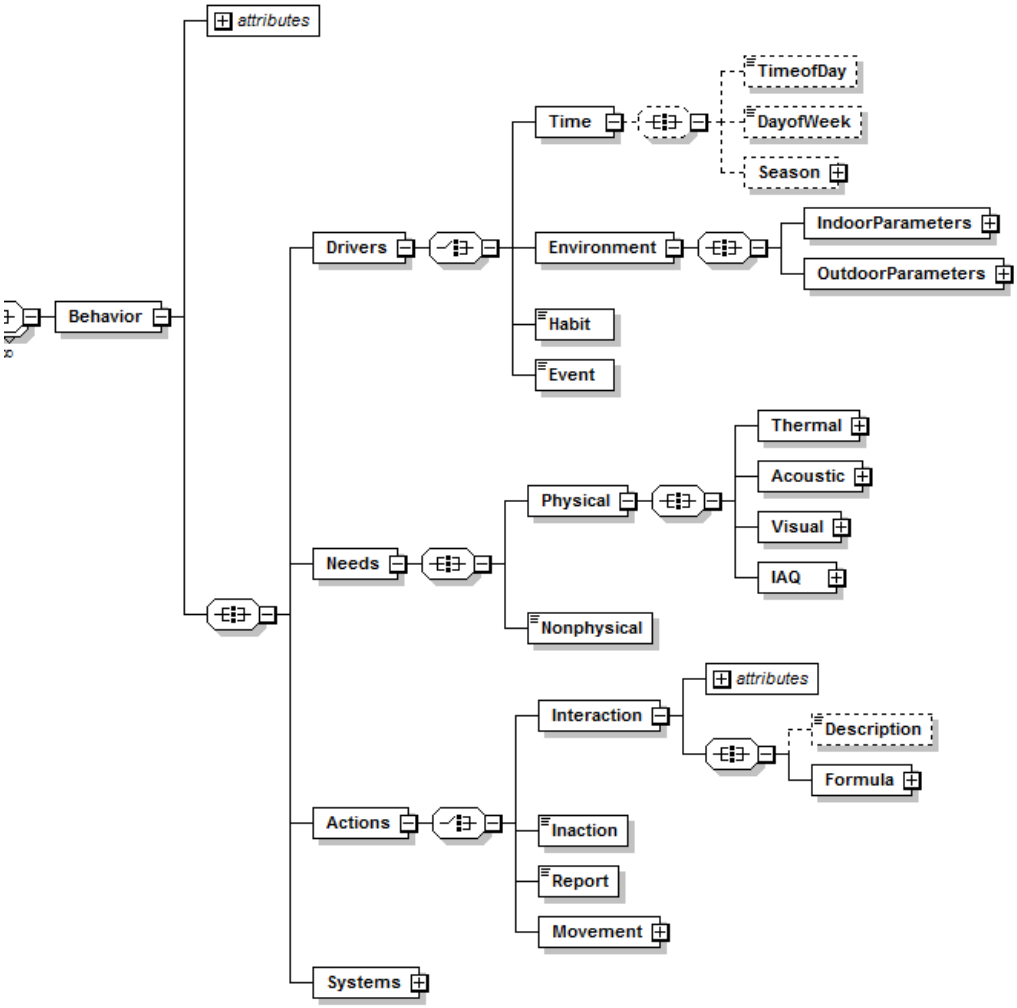

Figure 5 An overview of the main Behavior class of the obXML schema that implements the DNAS framework. (source: Yan et al. 2017)

BuildingSync

BuildingSync is a schema focused on energy audit data for commercial buildings. It was developed to enable exchange of such data between different software and databases involved in auditing process.

It standardizes and aggregates heterogeneous data into a common format. Thus, energy audit data represented in a standard, consistent and comparable format, facilitates not only seamless data exchange between auditing tools, but when aggregated - can enable potentially significant large-scale analysis (DeGraw et al. 2018). The schema has been recently upgraded to include data relevant for creation of physic-based energy models of a building or building-groups from data collected during an audit. Such models can be utilized with simulation software for performance assessment in retrofitting scenarios.

This section presented the spectrum of relevant building domain ontologies pertaining to various thematic categories. The Building Performance Data ontology described in the following chapter puts emphasis on performance data (such as performance measures and indicators) that spans through many categorical domains and therefore requires a comprehensive metadata schema to accurately capture the complexity of their nature.

1.3. Overview

The details of the contents of the remaining chapters are given below:

- **Chapter 2: Building Performance Data Ontology**
  This chapter presents details on the proposed BPD ontology and its implementation process. It includes an overview of building performance related data and its categorical domains, presents the proposed data schema for building performance data, describes steps in the workflow of transformation of various data sources to ontologically structured performance data. Finally, it exemplifies the ontology implementation process on a large set of performance data.

- **Chapter 3: Demonstration of Applications**
  This chapter presents examples of application of ontologically structured data in a series of basic to advanced building performance analysis and assessment scenarios. This includes: data retrieval, data visualization, interfacing to advanced performance analysis tools, ontology driven web-based PV system performance assessment tool and multi-domain ontology driven PV performance analysis.

- **Chapter 4: Conclusion**
  This chapter provides an overview of the presented work discusses selected challenges and gives an outlook on future research directions.

Please note that the chapters contain excerpts from the work published by the author in the course of research progress. This includes: Mahdavi and Wolosiuk (2019a, 2019b, 2021), Mahdavi et al. (2021), Wolosiuk and Mahdavi (2020a, 2020b), Wolosiuk et al. (2021).

CHAPTER 2

Building Performance Data Ontology

2.1. Ontology for building performance data

Peter de Wilde in his book "Building Performance Analysis" (De Wilde 2018) defines building performance as:

> "Building performance is a concept that describes, in a quantifiable way, how well a building and its systems provide the tasks and functions expected of that building. Requirements may stem from three main views: an engineering view of buildings as an object, a process view of building as a construction activity, and an aesthetic view concerned with the notions of form and appreciation. Important performance requirements in the engineering view pertain to building quality, resource savings, workload capacity, timeliness and readiness."

Building performance is usually verified (quantified) through the means of testing, calculation or combination of the two (Foliente 2000). In this context testing denotes collection of various measurement data as for example that collected by sensors and devices present in a building or via direct measurement methods. Such measured data could be, but not necessarily is, a performance measure itself. It can potentially be a variable that contribute to a compound performance indicator, obtained through calculation or

simulation.

Performance simulation can combine testing (measuring) and calculation as it makes use of both empirical (e.g. measured) and theoretical data (e.g. generated, modeled or standard values). Performance analysis can be used to improve buildings and the built environment. Two of the main phases of the building's life cycle, namely the design and the operational phase can particularly benefit from assessment procedures. In the design phase, assessment operations (e.g. computational simulations) can help measure impact of architectural design decisions on variety of aspects of building performance (e.g. energy efficiency, occupant's thermal, visual or acoustical comfort) and suggest alternative design solutions. During the operation phase, performance can be monitored and assessed for deterioration levels or help in faults and errors detection. Finally, the performance analysis can support enhancement or reinstatement of building's performance levels during the refurbishment process (De Wilde 2018).

Performance Data

A performance assessment of both building designs as well as existing buildings relies on extensive amounts of monitored or calculated data on buildings' behavior to derive the values of key Building Performance Indicators (BPIs) and performance measures. Both primary performance data (i.e., sensor data or simulated data) and high-level building performance indicator values come in various forms, degrees of resolution, and application domains.

An efficient and effective processing of such heterogeneous information base could benefit greatly from a well-structured ontological schema that would cover the multiple levels of complexity involved. Such ontology is essential for scientific community to facilitate the aforementioned mentioned performance data analysis procedures towards building design, operation, and retrofit optimization throughout the buildings' life cycle. It would also provide a solid basis for development of generic tools and applications such as interfaces to established modeling tools, visualization engines that could further support optimization or Building Management Systems (BMS) applications and provide deeper insight into the data.

The present work includes research efforts to develop and test a comprehensive Building Performance Data (BPD) ontology and associated schema. The BPD ontology builds upon foundational ontological work on monitoring data (see Mahdavi et al. 2016,2017,2018) and recent efforts to form an explicit ontology for building performance indicators (see Mahdavi and Taheri 2018; Mahdavi and Wolosiuk 2019a, 2019b, 2021; Wolosiuk and Mahdavi 2020a, 2020b).

The proposed ontology relies on an extensive review of common performance indicators as well as other performance related data originating from monitoring systems, simulation applications or models. The developed schema captures the necessary attributes concerning

performance data and indicators from a diverse spectrum of performance categories.

Building Performance Indicators

To provide a robust classification framework for representation of performance data in a robust schema a number of categories must be recognized both in a domain of performance indicators as well as performance measures. BPIs pertain to multiple categorical domains that assess performance related to building systems, equipment or envelope and equally that related to "habitability" aspects of the building, including human health, comfort, and satisfaction (Mahdavi 1998, 2011).

There are many performance indicators in use and a list of such is constantly modified and new positions are being added. In consequence a comprehensive review of all available indicators is difficult. Nevertheless, the majority of commonly known and used BPIs can be compiled, reviewed and categorized. Such effort by Constantino (2017) organizes reviewed indicators in 5 thematic domains, including energy efficiency, hygro-thermal performance, thermal comfort, air quality, visual environment and acoustical environment. These classes are primarily related to indoor environmental factors as well as energy- related building performance variables.

Of course, the building performance spectrum is not limited to these categories. Many other indicators related to building integrity, structure, safety and security, management, ecology or economy have been described in multiple standards and literature. Following the mentioned BPIs review, a general performance indicator classes taxonomy was proposed. Figure 6 presents an overview of the main performance categories (or classes), followed by sub-categories and instances of performance variables (indicators and measures).

Monitoring Data and Building Performance

Currently, in majority of public buildings, but also more commonly in private housing, a number of performance variables are directly monitored by the building systems or sensor networks. These monitored variables can be stand-alone performance measures or other performance-relevant data that are part of overarching compound indicator (this observation is discussed more in the following).

In order to be able to develop a comprehensive ontological schema for building performance data, identifying categories in which performance relevant monitoring data occurs (similarly to performance indicators) was a necessary step. Such classification was suggested in effort to create an ontology for building monitoring data (Mahdavi & Taheri 2017). The proposed classification framework was based on the related prior efforts (Mahdavi et al. 2005, 2016; Mahdavi 2011a 2011b; Zach et al. 2012).

Six main categorical classes have been proposed to provide an effective taxonomic classification to accommodate data streams related to monitoring systems. Namely: inhabitants, indoor environmental conditions, external environmental conditions, control systems and devices, equipment, and energy flows. Figure 7 pictures an overview of these

classes, with their suggested sub-classes and illustrative examples of corresponding monitored variables. Refer to Mahdavi and Taheri (2017) for reflections on selected categorical groups.

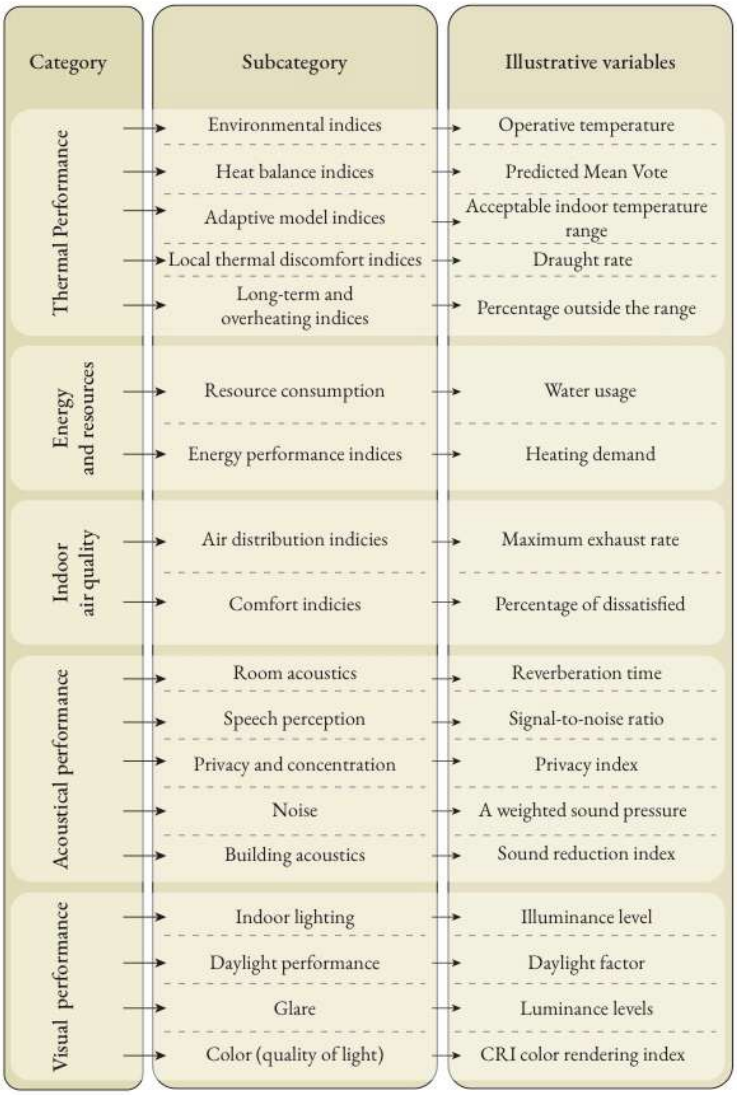

Figure 6 An overview of the five building performance categorical domains with their subsequent subcategories and illustrative instances (see Constantinou 2017 for more details).

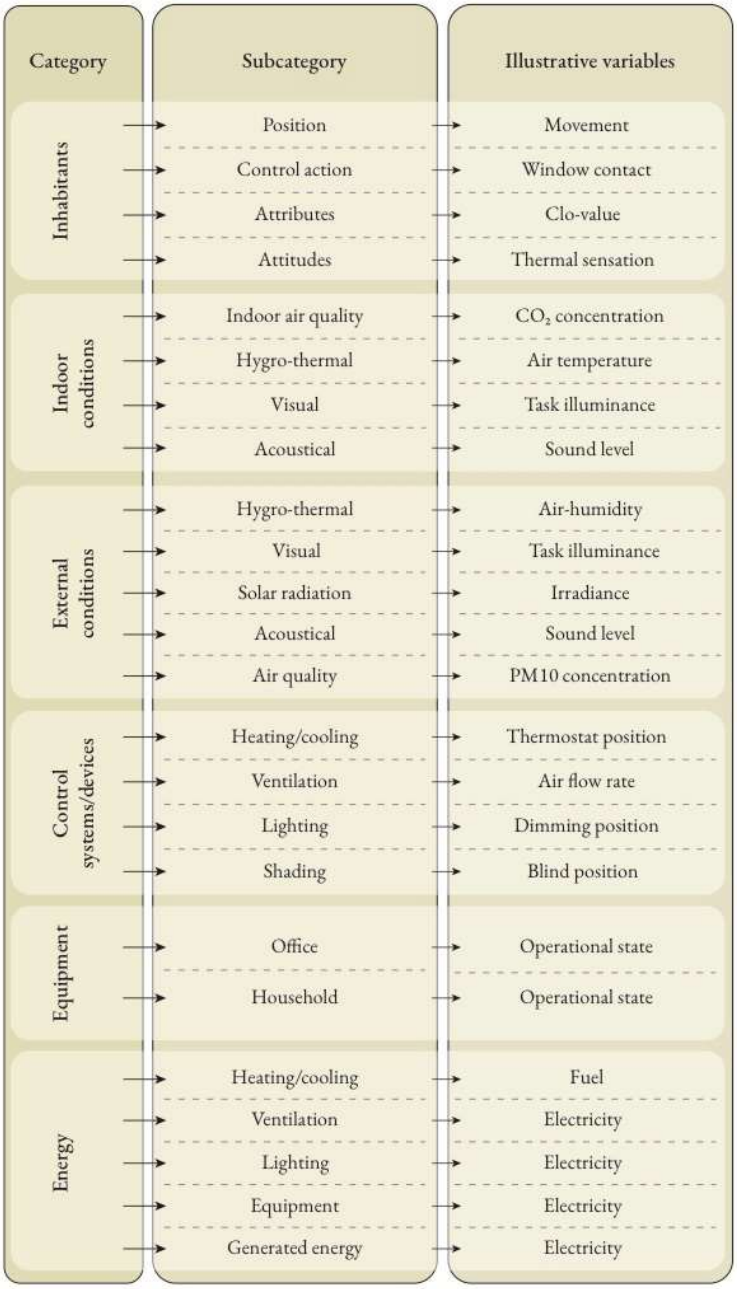

Figure 7 An overview of the six building monitoring data categorical domains with their subsequent subcategories and illustrative instances (modified based on Mahdavi and Taheri 2017).

Towards universal schema for performance data

Both of the aforementioned reviews were carried out in order to gain an overview of the respective subject matter, an insight into specification of variables pertaining to each of the classification groups and ultimately to form ontological schemas.

These insights helped compose a list of relevant metadata required to enable detailed attribution of performance indicators and monitored data originating from diverse categorical domains. Distinctive features of some categories and associated variables had to be reflected in the final schema through suitable properties. For example, in the case of the more complex schema for BPIs, many of the selected attributes are of a general and/or informative nature, and some are very specific, however required, to fully describe a narrow group of performance variables. Additionally, the relevance of some of the included attributes in relation to some more complex indicators may not have been apparent at first, but is critical to a full understanding of any indicator they describe and its associated value/quantity. Especially when the data is to be shared or re-used. Specifically, this group of attributes is associated with technical parameters of the method that was used to derive a certain indicator value.

Over the course of development of the more recent BPI ontology, it became clear that ultimately most performance indicators are compounds of measured data, or they can be traced back to a single measured variable. This observation indicated that attributes used to describe monitored data can be largely covered by the attribute scope of BPI data. For example, measured indoor temperature or CO_2 concentration level are primary variables (or measures) and can function as BPIs in the categories of thermal performance and indoor air quality respectively.

Other statistical treatment of such variable (e.g. values aggregation over time or space) can potentially lead to more complex indicator definitions. Furthermore, compound BPIs can be derived from multiple measured (also calculated or simulated) variables. For instance, daylight factor (an indicator in the visual performance domain) is derived by dividing measured or computed indoor illuminance by the outdoor horizontal illuminance obtained at the same point in time. In another example, the Predicted Mean Vote (Fanger 1970), an indicator of human sensation of thermal comfort, is calculated based on a number of measurable environmental variables (air flow speed, air temperature, radiant temperature, air relative humidity) and occupants' personal factors (degree of insulation provided by clothing – CLO value, metabolic rate due to actions). It was therefore logical to slightly modify the elaborate building performance indicators schema to include monitored data, arriving thus at a comprehensive Building Performance Data ontology that would encompass both previously mentioned ontologies (Mahdavi and Wolosiuk 2019b).

The structure of general schema for BPD is largely similar to that of the BPI proposed in Mahdavi and Taheri (2018). Table 1 presents the main features of the BPD schema. Each

variable considered falls into a specific main performance category and sub-category, and has a name (see Figure 6 and Figure 7 for examples). Instances of the same variable type require a unique identifier for different functional and relational purposes. Given a specific time and space each variable instance can assume a specific value. Each value can have a number of assigned properties and attributes. The variable's type primarily suggests whether the data classifies as quantitative or qualitative. If relevant, a level of measurement (nominal, ordinal, interval or ratio) can be added alongside data class. Such can render helpful in determining meaningful statistical operations or interpretation of the quantity. Quantitative data should be supplemented with the magnitude (expressed as a value or number), in case of vector-type variables also with a direction and a relevant unit for valid processing and interpretation.

Depending on the category of the variable, a number of additional properties in spatial, temporal and frequency domains can be specified. Spatial domain properties allow to associate a variable to a specific point in Cartesian coordination system, possibly a defined plane, volume or to a topologically relevant location (e.g. tagged room). If the magnitude of the variable was derived based on the integration of the number of data points over a space, the aggregation method (e.g. sum, average, median) and aggregation grid size can be specified.

Temporal domain properties are expressed in the schema via a time stamp (e.g. per instance of the sensor reading). Duration denotes the overall time frame to which a given variable value corresponds (e.g. daily, monthly, annual values). A time step denotes recurring time intervals to which measurement or simulation data have been assigned. Otherwise, the time step (similar to the grid size attribute) can specify the discretization resolution of pertinent temporal (spatial in case of the grid size) continua. Finally, if applicable the aggregation method of data over the time domain may have to be specified.

For a group of variables whose values belong to phenomena with wave character (e.g. sound, light, radiation), a group of attributes was defined in the frequency domain. Frequency range and/or band may be required to specify the analysis setup details (e.g. related to measuring equipment parameters) or pre-defined range of interest (such as human auditory system capability). Furthermore, the weighting (e.g. A-weighting of sound pressure levels in acoustics) and aggregation method can be specified.

As for the remaining variable property groups, if relevant, an agent ID attribute can be used to assign a given variable to a specific entity. Notes property group pertains to additional relevant information, for instance, on the indicators' derivation background. In case of a monitored variable, notes could include sources of data pointing to a unique ID of a measuring device or sensor. In case of derived and compound indicators, notes could further point to the corresponding computational procedure, such as applicable formulas, algorithms, and associated links and resources.

Table 1 General BPD schema (modified based on Mahdavi 2018)

Category	Name		
Sub-category	Name		
Variable	Name		
	ID		
	Value	Type	
		Magnitude (size)	
		Direction (vector)	
		Unit	
		Spatial domain	Point
			Plane
			Volume
			Topological reference
			Aggregation method
			Grid size
		Temporal domain	Time stamp
			Duration
			Time step
			Aggregation method
		Frequency domain	Range
			Band (filter)
			Weighting
			Aggregation method
	Agent		ID
	Notes	**Data sources**	Category
			ID
		Derivation method	Details (formula, link, etc.)

To illustrate the functionality and potential of the ontology, Table 2 includes three exemplary variables from different performance domains. Thereby, categories, subcategories, and relevant attributes of the selected variables are captured.

Table 2 Illustrative representation of three exemplary BPD variables following the structure of the proposed schema.

Category				Energy and resources	Thermal performance	Occupants
Subcategory				Energy performance indicator	Environmental indicator	Control action
Variable	Name			Heating Load	Air flow velocity	Window contact
	ID			AnnHL_BA_1807	AirFlo_R1_1808	WinCon_R1_W1
	Value	Type		Quantitative	Quantitative	Quantitative
		Magnitude		46	0.25	0
		Direction		-	[0 0 -1]	-
		Unit		kWh.m^{-2}	m.s^{-1}	-
		Spatial domain	Point	-	[1.50, 2.00, 1.10]	[1.00, 0.00, 1.20]
			Plane	-	-	-
			Volume	Building A	-	-
			Topo. Ref.	Building A	Room R_1	Room R_1
			Aggregation	-	-	-
			Grid size	-	0.20 m	-
		Temporal domain	Time stamp	-	08.05.18 10:30:00	08.05.18 09:20:00
			Duration	Annual	-	-
			Time step	1 hour	-	5 min
			Aggregation	Arithmetic sum	-	-
		Frequency domain	Range	-	-	-
			Band	-	-	-
			Weighting	-	-	-
			Aggregation	-	-	-
	Agent		ID	-	-	Occupant_1
	Notes	Data Sources	Category	Simulation	Simulation	Sensor
			ID	Sim_20180729_1	Sim_20180823_3	Con_15
		Derivation method		-	-	-

2.2. From heterogeneous data streams to application

Data or datasets (of a quantitative nature) consisting solely of an array of numbers without any additional descriptors cannot meet the requirements of the downstream applications. As descriptive and contextual content is added, data usability and scientific potential increase. Description of content, context, and structure should be an integral part of any dataset.

Files that contain data in tabular form (e.g. in CSV file format) are usually accompanied by a description file, or these details are stored within a file (e.g. in the header). This is not a very effective way to contextualize data, especially when it is a part of a larger heterogeneous data repository. The information stored in this form is inherently "flat". The bulk of context and the main hierarchy remain only on a descriptive level and must be interpreted.

In this context one of the primary functions of ontological data processing is to help achieve an explicit description of content and semantic integration of such heterogeneous data sources. Considering building performance data, this integration should not only provide a comprehensive overview of building's performance metrics for analysis purposes, but also allow for data utilization and interoperability across range of applications.

There might not be a single, universal approach to ontology implementation for data integration. However, certain set of operations are expected when processing quantitative data. Depending on a domain or a field of interest that an ontology pertains to, other operations may need to be recognized and applied. As a consequence, the implementation process may have to address a list of specific challenges that should be reflected in the workflow. Some of the relevant implementation questions are as follows:

- What type of data is to be semantically enriched and where does the data originate from?
- Should the data be preprocessed in any way?
- What relevant attributes are available or should be provided?
- How can attributes be attached to the data efficiently in accordance to the schema structure?
- How to store the ontologized data?

In the context of the presented effort, the answers to these questions set a foundation for creating workflow for transformation of heterogeneous performance data to ontologically structured information that is ready to be used in building performance assessment applications and beyond. Figure 8 presents schematic overview of the proposed structured process in which building-related measured or simulated "raw data" are processed and ontologized toward subsequent utilization in various applications. The different phases of this process are discussed in detail in the following sections.

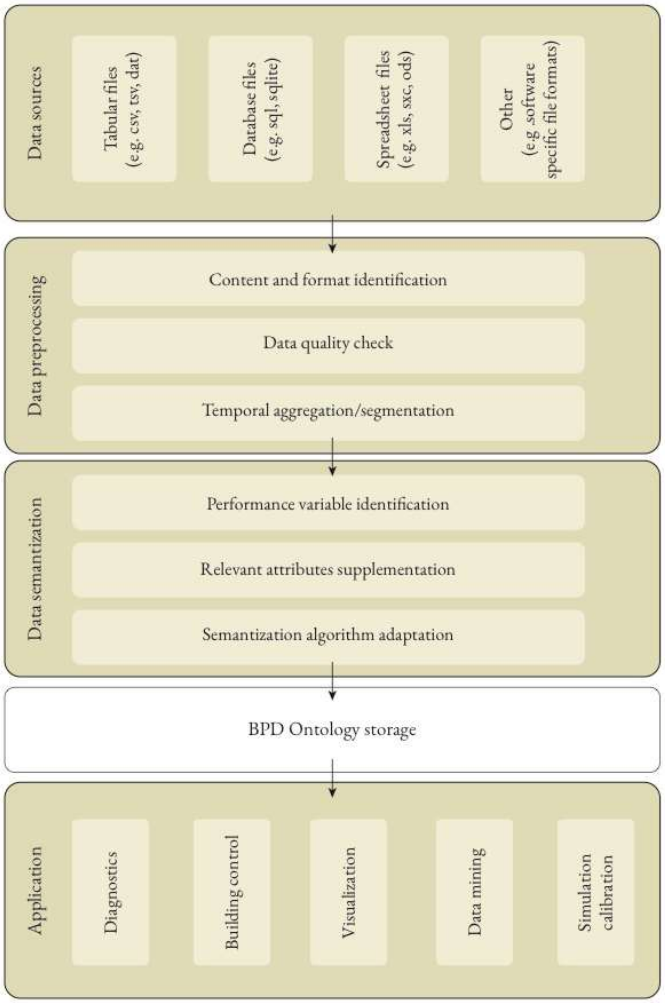

Figure 8 Schematic process overview for transformation (preprocessing, semantization, storage) of performance-data for use in various downstream applications (modified based on Mahdavi and Wolosiuk 2019b)

Data sources

Building performance data (BPD) originate from various heterogeneous sources. One of the most common BPD sources are building management and control systems, metering devices, or sensors providing information on indoor and outdoor environmental conditions. Likewise, computational building simulation applications can generate relevant performance data concerning the thermal, visual or acoustic conditions in the building. In addition, "virtual" sensors can deliver data that is derived using both simulated and measured values of pertinent building performance variables.

Due to very different data sources and the multitude of hardware and software components (e.g., sensors and data logging devices) and corresponding specifications, there is a large diversity in the ways raw data is acquired and stored. The acquisition and storage procedure can be influenced by specific manufacturer standards or even by individual requirements of a client or building manager. In case of data sources acquired from simulation, specific output formats are prescribed to the software.

This multiplicity of possible formats largely impacts the rest of the ontologization process, as data from different sources may require preprocessing in order to achieve a common standard regarding the temporal nature of the data (i.e. event-based recordings versus regular interval data) or its structural attributes. Some of the common output formats applied to building performance data include: i) delimited text files containing tabular data (e.g., CSV); ii) database files (e.g., MySQL); iii) spreadsheet files (e.g., XLS); or iv) other formats (e.g. software specific output formats). In the process of ontological data structuring, sources may need to be handled individually. For instance, semantically different and heterogeneously structured data can be stored in the same output format. This issue is discussed in the following section.

Data Preprocessing

Depending on the nature of data source and/or future application, methods of data preprocessing should be considered before supplementing properties and storing values of performance variables in a repository. The need for preprocessing is primarily related to the variability of data source types, volume and quality of data being integrated.

In this context, datasets (usually time series of measured performance values) should be examined in terms of data point sampling. There are two common sampling strategies for data recording, namely event-based and frequency-based (also called interval-based). With the event-based method, a measured value is only recorded if it indicates a meaningful change in the observed variable (e.g., indoor air temperature change above sensor's accuracy or above fixed threshold value) or there is a change in state of a device (e.g., open/close window). This type of sampling is increasingly present due to the growing number of wireless and low energy monitoring solutions and the need for an energy-efficient monitoring regime.

The frequency sampling method is based on a fixed time period between recordings of measured values. This type of sampling may seem more straightforward, nonetheless the sampling frequency can vary from device to device or from data source to data source.

For this reason, both methods face the following (application related) data usability challenge. Namely, many analysis scenarios involving data streams from multiple sources require a common temporal base (i.e. uniform intervals) for relevant data analysis operations. Instance of such operations include, for example, correlation analysis, data aggregation, or multi-variate statistical investigations. Depending on the application, different temporal resolution levels of data samples might be required. Theoretically, the temporal resolution could be kept at a very high level However, this can lead to data redundancy and unnecessarily high computational loads especially when processing large sets of data. Therefore, it might be appropriate to follow common practices in a domain and/or application scenario when deciding about the desired level of data resolution. For example, in case of typical analysis scenarios involving indoor thermal environment, 15-minute time intervals have been found sufficient. In contrast, high-quality monitoring of electrical power can require much shorter time intervals due to rapid spikes in the measured variable that would otherwise remain undetected.

The process of sensor-based data collection is prone to errors. Data quality control is a common practice that involves detection of flaws in a dataset to assure its validity and usefulness in future applications. There are a few common quality related challenges concerning data streams from monitoring devices. For instance, outliers are anomalous data points in a population of observations. Approaches to detect and mitigate outliers span from empirical methods (based on value range limits) to those rooted in descriptive statistics (e.g., Tukey's fences test for detection, or moving median for outlier removal).

Discontinuities in measurement records, stored in data repositories (also called - data gaps) represent another common problem. There are several strategies for filling in missing data points. In case of small data gaps resulting from the removal of outliers, the commonly used method is the previously mentioned moving median or mean. In case of larger data gaps, a number of data interpolation methods can be applied. The source of such quality issues might be related to sensor malfunctions, temporal power supply interruptions, voltage drops due to a discharging battery, errors in communication with logger or gateway, and other external events.

All of the mentioned preprocessing elements need to be considered when preparing ontologized data for storage, especially when data is to be shared or archived. Hidden flaws and inconsistencies in datasets that are not detected and rectified during the preprocessing phase, might hinder correct utilization of data by future users.

Data semantization

The purpose of implementing an ontology-based schema is to give data a meaning and

context. The ontology that is based on empirical study of a domain (or a field of interest), outlines the classification taxonomy, naming convention, and collection of applicable attributes.

As with many other cases, also in the case of BPD, the properties of data collections pertaining to performance variables or indicators need to be identified individually. This could become a cumbersome task that potentially involves manual processing of multiple variables, each requiring multiple properties to be identified. Depending on the case at hand, some properties might be available for scraping from the output files of the data sources (e.g., file name, header's content, column name, etc.). Nonetheless, in such an instance, strategies for information extraction need to be developed, resulting in additional efforts. This could be beneficial, but only when processing well-structured datasets.

Building performance data, particularly data coming from monitoring systems, often lacks in particular categorical or spatial details. These details need to be supplied by a system designer or individually obtained by an implementing expert. As such, the process of assigning relevant properties to data might be prone to human error, especially given the inherent diversity of diverse building performance data.

Ontology storage

Enriched building performance data must be serialized in a format that meets several requirements characteristic for ontologies in general, as well as particular requirements pertaining to a specific domain. A data model and its serialization format must be able to map the structure of hierarchical categories and their relations as specified in an ontology definition. It should also allow for assigning properties (detailed in schema) to categories, subcategories, and variables.

What characterizes the BPD ontology is that it typically applies to large sets of time-oriented data. Ideally the storage format must not only allow for an effective content mapping, filtering and access to categorical data attributes, but also – and more importantly – support efficient queries concerning time-dependent values of relevant variables. This condition excludes some of the simple text-based serialization solutions (due to serious performance issues) and requires some form of data base incorporation or data-specific file format utilization.

2.3. Implementation

Following the proposed structured workflow, the robustness of the BPD ontology was tested using a collection of diverse heterogeneous data sources. The dataset used for testing is a collection high-resolution measurement data points that come from multiple sensors that monitor occupants, equipment and the indoor environment in several office spaces in a university building (TU Wien) in Vienna, Austria. Moreover, the dataset also includes data

concerning external environmental conditions that were collected locally by multiple measuring instruments. Table 3 lists most of the relevant monitored variables in various categorical domains that are part of this dataset.

To put some of this data into a spatial context Figure 9 presents a plan of one of the office spaces with marked positions of the monitoring sensors (air temperature, relative humidity, air quality, presence, equipment and light electricity meters) and their original name-tags given during the initial monitoring system development. Not marked in the image are

Table 3 An overview of the performance variables available in the dataset that was used in the implementation.

Category	**Subcategory**	**Monitored variable**
Indoor conditions	Hygro-thermal conditions	Air temperature
		Relative humidity
	Visual	Overhead illuminance
	Indoor air quality	CO2 concentration
		Volatile Organic Compounds
External conditions	Hygro-thermal conditions	Air temperature
		Relative humidity
		Precipitation
	Daylight	Global Horizontal Irradiance
		Diffuse Horizontal Irradiance
		Sky Luminance
		Sky Radiance
		Sun Presence
	Outdoor air quality	CO2 concentration
	Outdoor air flow	Wind speed
		Wind direction
Energy and resources	Energy consumption	Active power consumption
		Energy meter reading
Control system and devices	Heating/Cooling system	Radiator surface temperature
Occupants	Position	Presence
	Control actions	Window contact

window contact sensors, radiator temperature sensors and less relevant hygro-thermal conditions sensors. Including sensors located in the remaining office spaces, there are a total of around 120 unique variables that capture indoor conditions and events.

The second group in this dataset are variables associated with external environmental conditions that are monitored locally using several high-grade monitoring instruments. Figure 10 shows a diagram with an overview of these monitored environmental variables (listed in table on the left side of the picture) in context of the corresponding monitoring system's architecture.

The implementation dataset in its majority is syntactically and structurally homogeneous. Syntactic homogeneity is achieved by using the same data model to store individual monitored variables. Based on the design approach of the monitoring system, each monitored variable is saved as a separate SQLite (Hipp, 2021) relational database file. Accordingly, each database file retains a structural homogeneity. Specifically, each file contains a two-column table with timestamps and a corresponding sensor reading. Figure 11 shows an example of the content of such a database file, in which sensor readings of the indoor air temperature are stored.

However, there are some examples of performance-related data in this dataset with a non-standard format or that differ syntactically and structurally and require individual handling. In the present implementation effort, these are the data sources concerning monitoring of the sky (see bottom left on Figure 10), such as those generated by the sky luminance camera or the sky-scanner. The data from the sky-scanner comes in a form of multiple tabular files (one csv file per each day of recording; see Figure 12 for an example) containing uniquely structured sequence of the measured values. These files need individual approach to pre-processing before being integrated, well supplemented with relevant information on the exact content and data points structure and stored in an ontologically consistent manner.

Another example of a non-standard data source in this dataset is output files generated by the sky luminance camera. These are high dynamic range images which essentially translate to a high-density matrix of precise luminance readings per measurement data point. Such can be additionally processed to a sequence of 145 values representing the hemispherical Tragenza (Tregenza 1987) sky matrix for use with advanced light simulation software (see Figure 22).

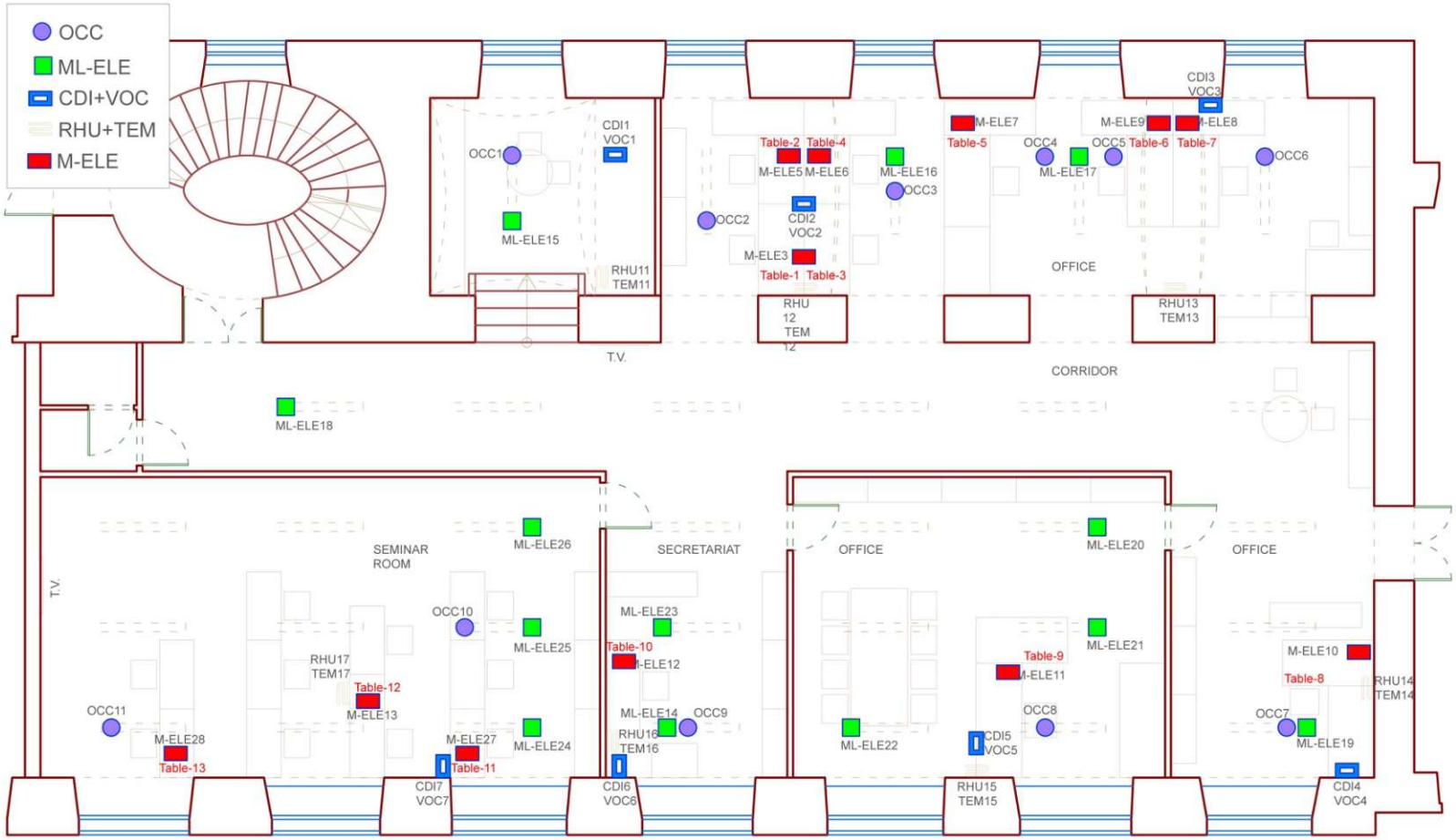

Figure 9 Office sensor locations plan.

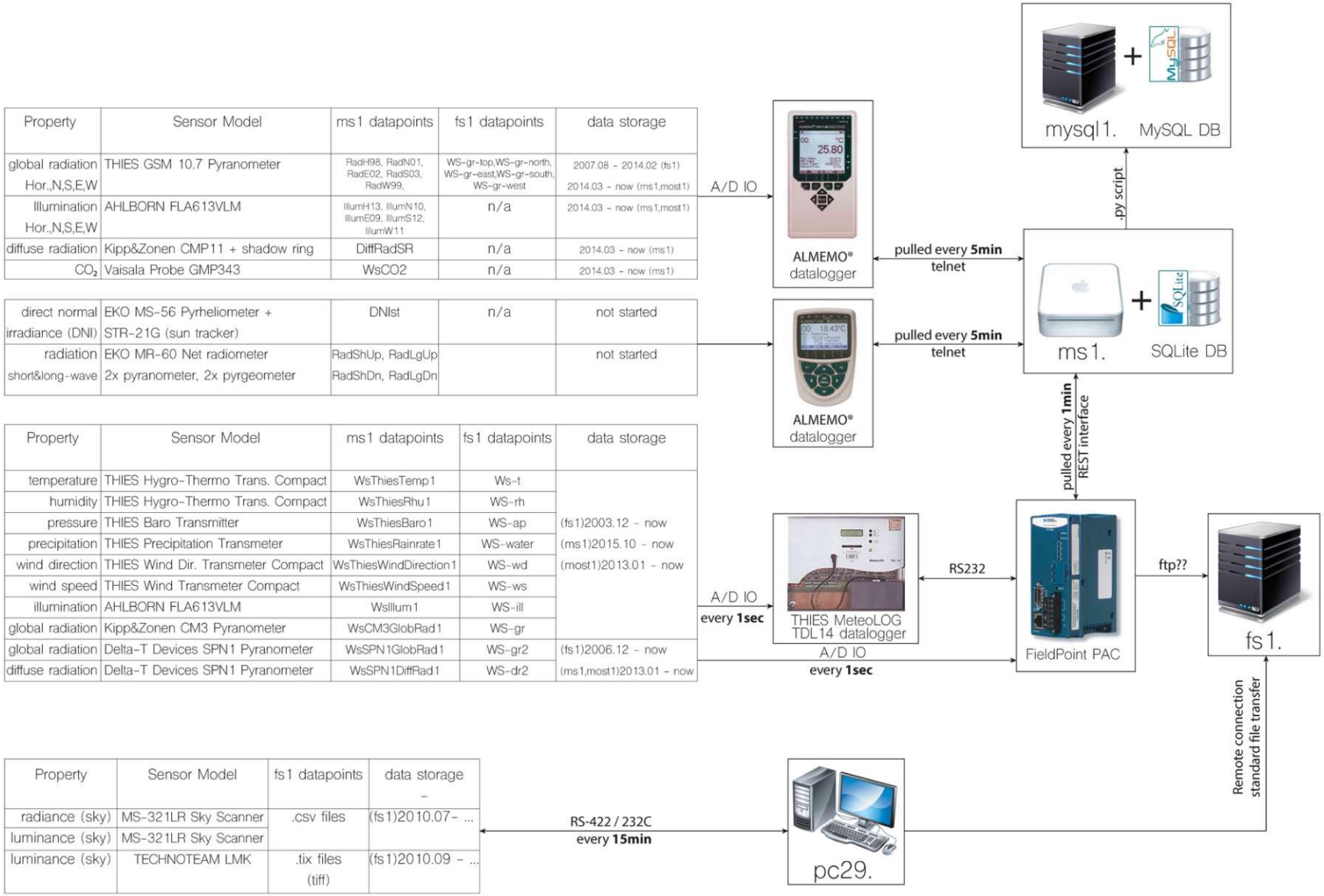

Figure 10 An overview of available monitored environmental variables and monitoring system architecture.

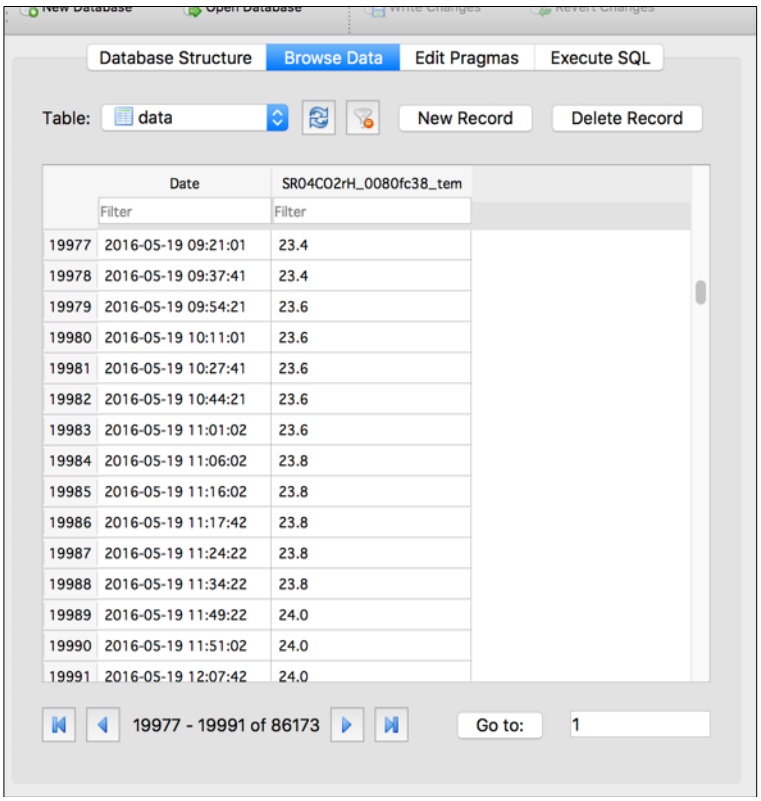

Figure 11 Sample content of a data source file used in the implementation. Two columns shown relate to the time stamp and the corresponding measured values of the indoor air temperature.

In summary, it can be said that the syntactic and schematic heterogeneity is only a technical issue. Different formats and representations of data sources can be adapted for available information extraction. Due to high variability of possible data sources and their implementation details, there is no single universal approach to this process. Individual approach means developing workflows, algorithms or tools to systematically extract the information which can be a cumbersome and time-consuming process.

We have established that the given implementation dataset is mostly homogenous on syntactic and schematic level, however it is heterogeneous on the semantic level. The content of a sole data file is (in the most cases) irrelevant both for humans and machines if a context is not provided with it. For example, information stored within said database files pertaining to any of the temperature measuring sensors (e.g. air temperature, radiator temperature) have the same meaning - namely the list of values stored in a file represent a measure of temperature. This information alone would be considered meaningless. Additionally, we cannot be certain of the unit and scale of the values in question. In the real-world, all of these data points have at least a certain spatial and/or temporal context.

Title	Sky Scanner MS-321LR Sky Luminance and Radiance distributions														
Firmware	1.10														
Date	2016/05/10														
Comment															
Longitude	+16 d 22.0 m														
Latitude	+48 d 12.0 m														
Timezone	UTC +1:00														
Unit	kcd/m^2	W/(m^2*sr)													
			1	2	3	4	5	6	7	8	9	10	11	12	...
L	05:00:00	05:04:34	0.75	0.80	0.87	0.94	1.00	1.04	0.93	0.91	0.95	0.91	0.75	0.66	...
R	05:00:00	05:04:34	8.30	8.94	9.97	10.84	11.76	12.48	11.27	11.79	11.40	11.00	9.14	8.01	...
L	05:15:00	05:19:34	0.49	0.50	0.55	0.62	0.66	0.69	0.60	0.51	0.56	0.59	0.55	0.58	...
R	05:15:00	05:19:34	6.29	5.73	6.55	7.77	8.35	9.06	8.25	7.85	7.84	8.15	7.50	7.23	...
L	05:30:00	05:34:34	0.89	0.88	0.89	0.92	0.86	0.82	0.66	0.50	0.50	0.45	0.42	0.49	...
R	05:30:00	05:34:34	12.38	12.59	12.64	13.63	12.37	11.15	9.15	7.35	6.07	5.56	5.66	6.89	...
L	05:45:00	05:49:34	1.22	1.16	1.15	1.18	1.23	1.25	1.01	0.82	0.75	0.81	0.80	0.76	...
R	05:45:00	05:49:34	17.01	15.97	16.10	16.37	17.52	18.22	14.75	12.75	9.70	10.79	11.16	10.35	...
L	06:00:00	06:04:34	1.93	1.62	1.58	1.58	1.53	1.59	1.57	1.43	1.34	1.21	1.24	1.53	...
R	06:00:00	06:04:34	27.23	22.08	21.69	22.24	21.12	22.32	23.41	23.66	17.67	15.66	16.90	21.00	...
L	06:15:00	06:19:35	3.13	2.78	2.58	2.61	2.88	3.14	2.83	2.68	3.39	3.11	2.52	2.43	...
R	06:15:00	06:19:35	42.02	37.94	33.35	32.27	35.32	42.47	40.76	42.19	44.43	39.96	35.06	34.53	...
L	06:30:00	06:34:35	3.20	2.89	2.96	2.98	3.19	3.34	3.27	3.21	4.29	3.66	3.12	2.85	...
R	06:30:00	06:34:35	35.26	33.82	35.91	34.12	37.76	40.71	39.91	45.42	49.58	42.59	39.20	35.06	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 12 Exemplary content of csv output file generated by sky-scanner control software.

In the given dataset, the temporal context is provided in the form of time stamps for each measured value, but the rest of the context is not disclosed. All other semantic details necessary for a proper interpretation of the information about the data source must be determined so that they can be provided later during the semantization part of the proposed workflow. Before semantic enrichment of the selected data, however, two preprocessing operations must be considered, namely the temporal aggregation/segmentation and the data quality check.

The designated dataset includes variables with both event-based and interval-based sampling records. The majority of indoor environment monitoring sensors used in this system are ultra-low power, battery-less wireless sensors that have to manage small amounts of power very efficiently. Therefore, a measured value is only transmitted when there is a significant change in the monitored variable, which leads to an event-based characteristic of the stored records (see Figure 11 for an example of event-based temperature measurements records from one of these sensors).

On the other hand, the variables related to the monitoring of the external environment are recorded periodically according to requested interval length. The selected intervals of 1 minute. (for hydro-thermal and air flow monitoring instruments), 5 minutes (for daylight

monitoring sensors) and 15 minutes (for sky monitoring sensors) are based on the monitoring system design decision (see Figure 10 for the system overview).

At this level of implementation process, there was no aggregation nor segmentation performed on any of the selected variables. The reason behind was to retain the original values recorded by the sensors for more flexibility at the application level. In fact, data aggregation was implemented as feature of a tool in one of the illustrated application examples in the following chapter. If necessary, after the ontological integration of the data set, new spatially or temporally aggregated or segmented variables can be generated from the existing ones.

The dataset in question was subjected to a basic data quality check. The characteristics of the dataset and related issues were previously recognized and known to the author. Thus, the quality check was not performed "formally" (e.g. no calculated nor interpolated data points were inserted), as this was not the main focus of the research. Nevertheless, certain quality control relevant actions were undertaken with regard tothe variables of interest. For example, due to previously recognized occasionally occurring server failures resulting in sensor-server communication interruption, an algorithm was developed to specifically detect quantitative anomalies in the monthly number of generated data points stored in a database. This particular analysis gave some insight on the time periods without any missing entries and allowed to select the best quality portion of the dataset for use in application testing. In addition, missing data treatment methods were implemented as a feature of the tool that is illustrated in one of the application examples.

Following the proposed workflow, in the next step of ontological data integration process, the dataset was given a semantic meaning by providing each of the selected variables with relevant attributes as per the proposed performance data schema. As discussed previously the only context available from within used data sources is temporal. The remaining context and semantic details had to be identified.

There are many potential sources of metadata and means to help identify it. Starting at a low level, such a source can be the name of a data file or the header of a data file (for text file sources), data column name or accompanying metadata table (for database sources). In this implementation, the names of the database files (e.g. "SRW01_00016c48_con_rawdata.db") indicate the measured performance variable, the sensor model name, and provide a unique physical address that can be used to link the file to other related documentation (see Figure 14 for more file name examples in the column A).

The potential availability or quality of documentation or similar sources with relevant information about the data varies from case to case and depends heavily on the design of the monitoring system or application as well as on implementation decisions. If data comes from a BMS system, it is very likely that all system details are well documented. If, on the other hand, a data source was a part of specific application instance or used in a peculiar

workflow, it may not originally have been designed for reuse, thus none or very few of the semantically relevant details might have been documented. In such case an insight from a designer or managing person is required.

To illustrate the potential sources of semantic attributes that a person implementing an ontological schema may need to work with to gather information, the following sources have been used to find spatial properties of some of the variables in the implemented dataset. Figure 13 previews content of a spreadsheet with the inventory of wireless sensors and related equipment. The most relevant information in this table is the physical address of the sensor and the unique ID of a monitored variable assigned to it. In this way we could cross reference database files with a specific ID of the monitored variable to extract semantic information from other sources.

For example, the office plan shown on the Figure 9 with exact spatial position of the sensors. This plan was used to extract relevant spatial domain attributes (e.g. position in Cartesian space, topological details) of some of the performance data variables. The rest of the relevant or required metadata that could not be scraped from the aforementioned sources were derived empirically or were based on expert knowledge and monitoring system insight.

Given the multitude of performance data variables in the implementation dataset, the attributes of which must be determined and processed in order to ultimately be serialized in a file, a specific solution was expected to mitigate this process. Therefore, to facilitate the attributes supplementation and deposition, all variables and their attributes were aggregated in a single csv file (see Figure 14). The content of this file follows specification of the BPD schema and is used to enable efficient integration and storage of the data via designed conversion algorithm. The data from the sky-scanner, since it is syntactically different from the rest of dataset, required individual approach and a separate set of conversion functions.

Python programming language was used in the process of performance data transformation. Several functions for extraction and conversion of the input data streams, attributes organization and supplementation, and structured storage in a HDF5 file were developed. The HDF5 (The HDF Group 2019) is a primarily scientific data format capable of storing various data objects, adapting high-level data schemas and executing very high-performance queries on large datasets. It allows grouping and organizing objects in a hierarchical manner, enables the assignment and accommodation of complex metadata to the elements of the structure and the linking of elements.

Given these features, the HDF5 file format appeared to provide a suitable foundation for testing the BPD ontology and the proposed data schema. Note that it is not suggested that this is the only or the ultimate implementation solution. Another implementation approach, for instance, could be based on Semantic Web (SW)technologies. Because in the application phase, the emphasis was on high-volume data processing (e.g. long-term visualizations),

selecting an all-in-one high-performance format seemed appropriate for this purpose. To achieve comparable level of data query performance in a SW based implementation, the use of solutions such as external data storage repositories would be required and would certainly add another layer of complexity in any application scenario. Figure 15 presents an exemplary overview of the content of HDF5 file with hierarchically structured building performance variables and their instances in different categorical groups.

	A	B	C	D	E	F	G
1	**sensor id**	**fieldbus**	**physical address**	**DB data point name**	**location**	**hardware description**	**notes**
10	con8	enocean	00016C88	con8	bpi		
11	con9	enocean	00016C36	con9	bpi		
12	con10	enocean	00016C44	con10	bpi		
31	con29	enocean	00016C60	con29	leh		
32	con30	enocean	00016C7B	con30	leh		
49	con47	enocean	0001afc0	con47			
50	con48	enocean	0001aebe	con48			
59	ele7	enocean	008051ab	ele-pow7, ele-met7			
60	ele8	enocean	00803f2a	ele-pow8, ele-met8			
61	ele9	enocean	00804a80	ele-pow9, ele-met9			
84	elec_32	enocean	0103408F		grieskirchen		25F4
85	elec_33	enocean	01033AB8		grieskirchen		25F5
171	swi1_swi2	enocean	0013d2e1	swi1, swi2			
172	swi3_swi4	enocean	0013cfd0	swi3, swi4			
173	occ1_lux1	enocean	00020AC0	occ1, lux1			
174	occ2_lux2	enocean	000308B4	occ2, lux2			
215	rhu3_tem3	enocean	00032DE9	rhu3, tem3			
216	rhu4_tem4	enocean	00032E3C	rhu4, tem4			
217	rhu5_tem5	enocean	00032F72	rhu5, tem5			
233	tcon2	enocean	00032F1D	tcon2		0-80°C, L2m, 6x50mm	PT1000
234	tcon3	enocean	00032E36	tcon3		0-80°C, L2m, 6x50mm	PT1000
259	cdi4	enocean	00033F3D	cdi4			
260	cdi5	enocean	00033F41	cdi5			
274	voc3	enocean	00033F45	voc3			
275	voc4	enocean	00033F09	voc4			
324	con53	enocean	0001BC38		grieskirchen		
325	con54	enocean	0001BC36		grieskirchen		
382	con72	enocean	0001AEF1		BPI-turm		
383	con73	enocean	0001AFB6		BPI-turm		

Figure 13 Preview of the inventory file concerning wireless sensors and equipment.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1										Spatial domain							Tempo
2	Filename	Category	Sub_Category	Variable	ID	Type	Magnitude	Direction	Unit	Point	Plane	Volume	Topological_reference	Aggregation_method	Grid_size	Time_stamp	Duration
3	SR04rH_00038e9a_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem14	quantitative			°C	0.8 11.5 1		TU Wien Hauptgebäude	BPI Open-Space NW				
4	SR04rH_00038e31_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem12	quantitative			°C	10.5 4.5 1.5		TU Wien Hauptgebäude	BPI Open-Space S				
5	SR04rH_00038ea3_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem13	quantitative			°C	4 4.5 1		TU Wien Hauptgebäude	BPI Open-Space S				
6	SR04rH_00038e4c_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem16	quantitative			°C	14 12.5 1		TU Wien Hauptgebäude	BPI Secretariat				
7	SR04rH_00038e44_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem15	quantitative			°C	7 11 1		TU Wien Hauptgebäude	BPI Office				
8	SR04rH_00038e2b_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem11	quantitative			°C	14.5 4.5 1		TU Wien Hauptgebäude	BPI Kitchen				
9	SR04CO2rH_0082124f_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem23_new	quantitative			°C	14.5 1.5 1		TU Wien Hauptgebäude	BPI Kitchen				
10	SR04CO2rH_0080fc38_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem_t-2_new	quantitative			°C	1 12.5 1		TU Wien Hauptgebäude	BPI Open-Space NW				
11	SR04CO2rH_0100a66c_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem__t-4_new	quantitative			°C	10.5 3 0.8		TU Wien Hauptgebäude	BPI Open-Space S				
12	SR04CO2rH_0100f863_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem20_new	quantitative			°C	13 12.5 1		TU Wien Hauptgebäude	BPI Secretariat				
13	SR04CO2rH_0081da83_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem21_new	quantitative			°C	20 12 1		TU Wien Hauptgebäude	BPI Seminar Room				
14	SR04CO2rH_008206d9_tem_rawdata.db	IndoorConditions	HygroThermalConditions	AirTemperature	tem22_new	quantitative			°C	4 3 1		TU Wien Hauptgebäude	BPI Open-Space S				
15																	
16	SR04rH_00038e9a_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu14	quantitative			%	0.8 11.5 1		TU Wien Hauptgebäude	BPI Open-Space NW				
17	SR04rH_00038e31_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu12	quantitative			%	10.5 4.5 1.5		TU Wien Hauptgebäude	BPI Open-Space S				
18	SR04rH_00038ea3_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu13	quantitative			%	4 4.5 1		TU Wien Hauptgebäude	BPI Open-Space S				
19	SR04rH_00038e4c_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu16	quantitative			%	14 12.5 1		TU Wien Hauptgebäude	BPI Secretariat				
20	SR04rH_00038e44_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu15	quantitative			%	7 11 1		TU Wien Hauptgebäude	BPI Office				
21	SR04rH_00038e2b_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu11	quantitative			%	14.5 4.5 1		TU Wien Hauptgebäude	BPI Kitchen				
22	SR04CO2rH_0082124f_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu23_new	quantitative			%	14.5 1.5 1		TU Wien Hauptgebäude	BPI Kitchen				
23	SR04CO2rH_0080fc38_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu_t-2_new	quantitative			%	1 12.5 1		TU Wien Hauptgebäude	BPI Open-Space NW				
24	SR04CO2rH_0100a66c_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu_t-4_new	quantitative			%	10.5 3 0.8		TU Wien Hauptgebäude	BPI Open-Space S				
25	SR04CO2rH_0100f863_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu20_new	quantitative			%	13 12.5 1		TU Wien Hauptgebäude	BPI Secretariat				
26	SR04CO2rH_0081da83_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu21_new	quantitative			%	20 12 1		TU Wien Hauptgebäude	BPI Seminar Room				
27	SR04CO2rH_008206d9_rhu_rawdata.db	IndoorConditions	IndoorAirQuality	RelativeHumidity	rhu22_new	quantitative			%	4 3 1		TU Wien Hauptgebäude	BPI Open-Space S				
28																	
29	SR04CO2rH_0080fc38_cdi_rawdata.db	IndoorConditions	IndoorAirQuality	IndoorCO2level	cdi_t-2_new	quantitative			ppm	1 12.5 1		TU Wien Hauptgebäude	BPI Open-Space NW				
30	SR04CO2rH_0100a66c_cdi_rawdata.db	IndoorConditions	IndoorAirQuality	IndoorCO2level	cdi_t-4_new	quantitative			ppm	10.5 3 0.8		TU Wien Hauptgebäude	BPI Open-Space S				
31	SR04CO2rH_0100f863_cdi_rawdata.db	IndoorConditions	IndoorAirQuality	IndoorCO2level	cdi20_new	quantitative			ppm	13 12.5 1		TU Wien Hauptgebäude	BPI Secretariat				
32	SR04CO2rH_0081da83_cdi_rawdata.db	IndoorConditions	IndoorAirQuality	IndoorCO2level	cdi21_new	quantitative			ppm	20 12 1		TU Wien Hauptgebäude	BPI Seminar Room				
33	SR04CO2rH_008206d9_cdi_rawdata.db	IndoorConditions	IndoorAirQuality	IndoorCO2level	cdi22_new	quantitative			ppm	4 3 1		TU Wien Hauptgebäude	BPI Open-Space S				
34	SR04CO2rH_0082124f_cdi_rawdata.db	IndoorConditions	IndoorAirQuality	IndoorCO2level	cdi23_new	quantitative			ppm	14.5 1.5 1		TU Wien Hauptgebäude	BPI Kitchen				
35																	
36	FWZ61_00803615_met_rawdata.db	EnergyAndResources	ResourceConsumption	ElectricEnergyConsumption	ele-met1	quantitative			kWh	17 2.5		TU Wien Hauptgebäude	BPI Kitchen				
37	FWZ61_00803890_met_rawdata.db	EnergyAndResources	ResourceConsumption	ElectricEnergyConsumption	ele-met2	quantitative			kWh	14.5 1.5		TU Wien Hauptgebäude	BPI Kitchen				
38	FWZ61_00803672_met_rawdata.db	EnergyAndResources	ResourceConsumption	ElectricEnergyConsumption	ele-met3	quantitative			kWh	11.5 4		TU Wien Hauptgebäude	BPI Open-Space S				
39	00803932	EnergyAndResources	ResourceConsumption	ElectricEnergyConsumption	ele-met4	quantitative			kWh	10 4		TU Wien Hauptgebäude	BPI Open-Space S				
40	FWZ61_008036b1_met_rawdata.db	EnergyAndResources	ResourceConsumption	ElectricEnergyConsumption	ele-met5	quantitative			kWh	11.5 2.5		TU Wien Hauptgebäude	BPI Open-Space S				
41	FWZ61_00803b09_met_rawdata.db	EnergyAndResources	ResourceConsumption	ElectricEnergyConsumption	ele-met6	quantitative			kWh	10 2.5		TU Wien Hauptgebäude	BPI Open-Space S				

Figure 14 Preview of the tabular file containing gathered details pertaining to performance data and variables.

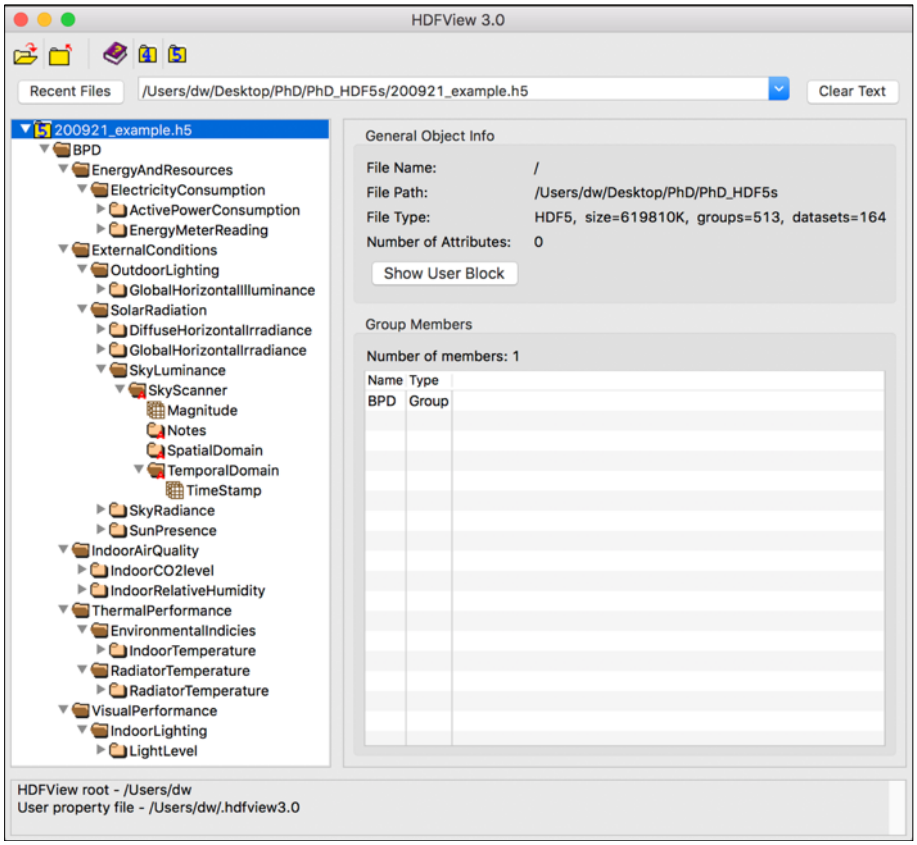

Figure 15 Content of a HDF5 file with ontologically structured performance data.

CHAPTER 3

Demonstration of application

3.1. Information retrieval

Note: The following section is based on and contains excerpts from the following, previously published conference papers - Mahdavi and Wolosiuk (2019a, 2019b).

The most basic functionality that is enabled by the ontology implementation is information retrieval. Information that is available in a semantically enriched dataset can be very precisely selected or extracted using logical queries. Any of the attributes present in the BPD schema might potentially be relevant in such queries. For example, we might be interested in limiting our selection to specific performance category or variable name, we can further limit the selection variables available in certain space, but also, we might be interested in those associated with a specific occupant. Depending on application, there are many different query scenarios and these queries are most likely the backbone of any advanced performance application.

As a part of initial BPD ontology testing, a series of algorithms were created in the Python environment to test querying efficiency of ontologically structured data. The main focus of this low-level application was to extract target variables that fulfill a specified combination of spatial, temporal, and categorical criteria. After successful extraction, the data of interest

was further processed in terms of descriptive statistics and data visualization (e.g., box plots, histograms, line plots).

To illustrate this process, consider the example of a two-step test query of the implemented dataset. In the first step the variables are filtered to ones that have "BPI Secretariat" as the "Topological reference" in their Spatial domain attributes group. To preview the content of the selected variables and demonstrate some visualization potential, a trend line for each of the selected variables was generated for a content preview in a selected time period (here 1-31.05.2016). Figure 16 shows a result of the executed algorithm in the PyCharm IDE - a

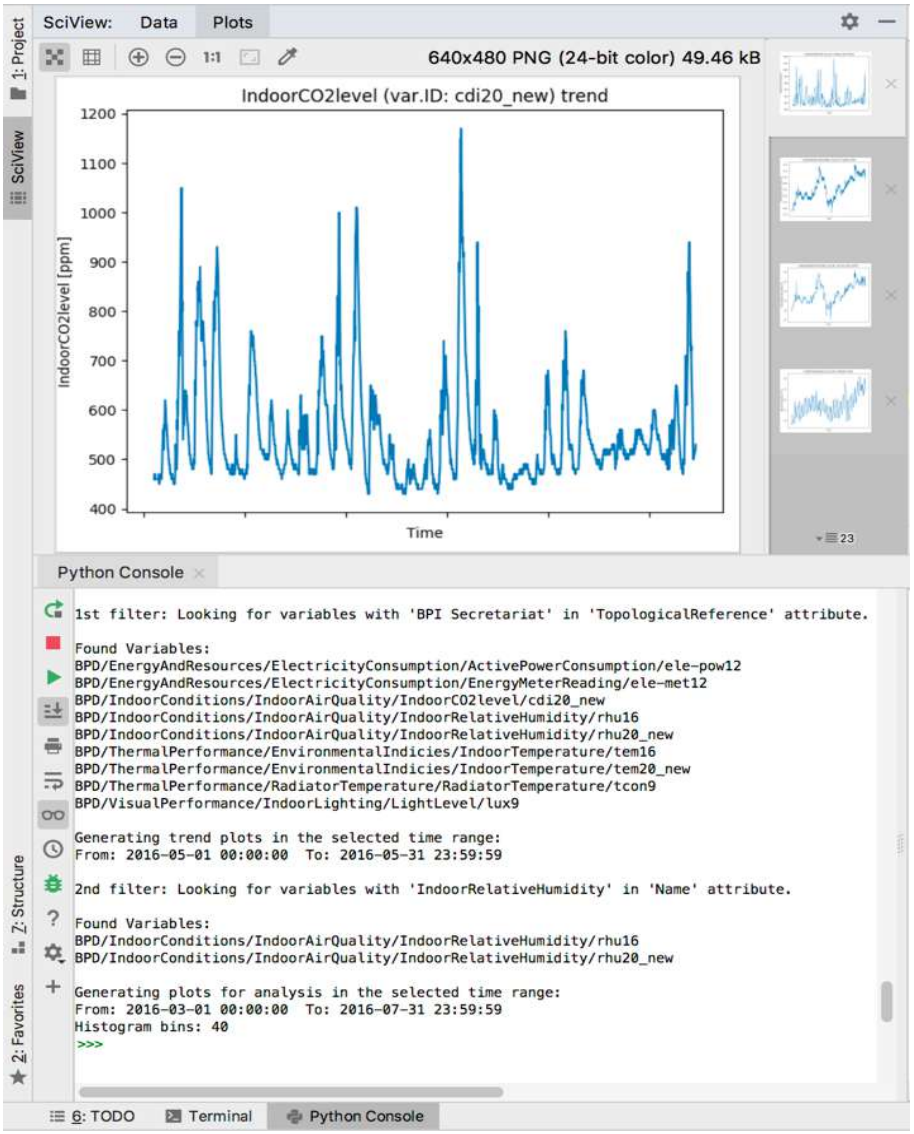

Figure 16 Information retrieval using python programming environment. Output of two consecutive logical queries visible in the bottom window, the upper window previews generated plots.

Python integrated development environment (JetBrains 2021). In the lower window, the variable instances found after the first query step are listed and a function is executed to generate the trend plots for all selected instances (see Figure 18).

The second query step is to extract only the instances of "Indoor Relative Humidity" variable that are available and generate additional statistically relevant visualizations. The final result of the query (variable instances "rhu16" and "rhu20_new") are visible in the bottom window on the Figure 16. Figure 19 presents the generated set of three visualizations for one of the selected variable instances in the selected time frame (1.03.2016 – 31.07.2016).

This example illustrates the utility and effectiveness of the well-formed semantically enriched data. Together with a simple user interface the presented algorithm could become part of a visualization/exploration tool.

As mentioned in the previous chapter, the HDF5 file format is very efficient in terms of query performance and can be used in different programming environments. To test and demonstrate this universal support, an algorithm was developed in MATLAB software to process queried data (hourly aggregation of event-based measured values) and generate an annual hourly tile map visualization.

For this purpose, logical queries were formulated to satisfy specific combinations of categorical, spatial, and temporal criteria. Specifically, "overhead illuminance level" as variable name, "seminar room" as topological reference, and "year 2017" as temporal constraint were defined. The values were aggregated to form an hourly based value matrix (see Figure 17)with an additional visual based information in form of color-coded value intervals.

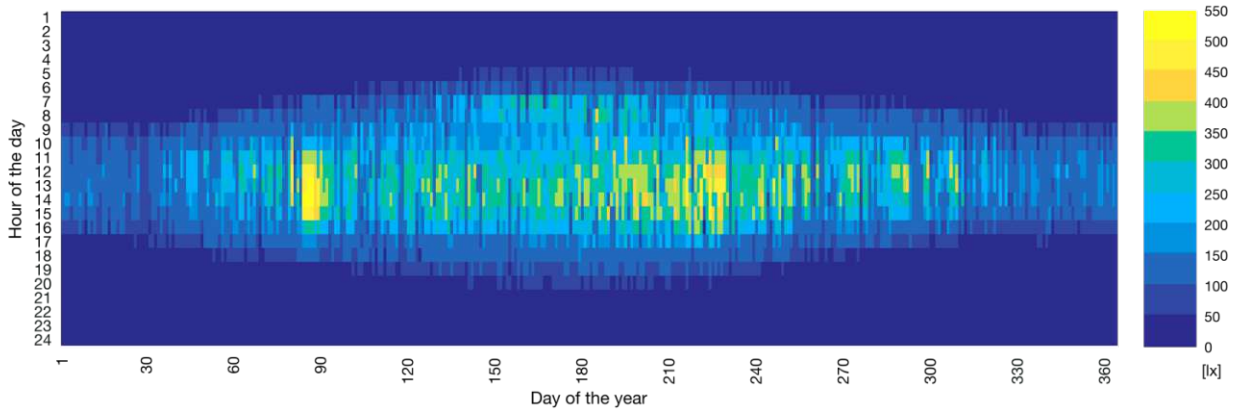

Figure 17 An annual tile map visualization of measured overhead illuminance levels in an office area.

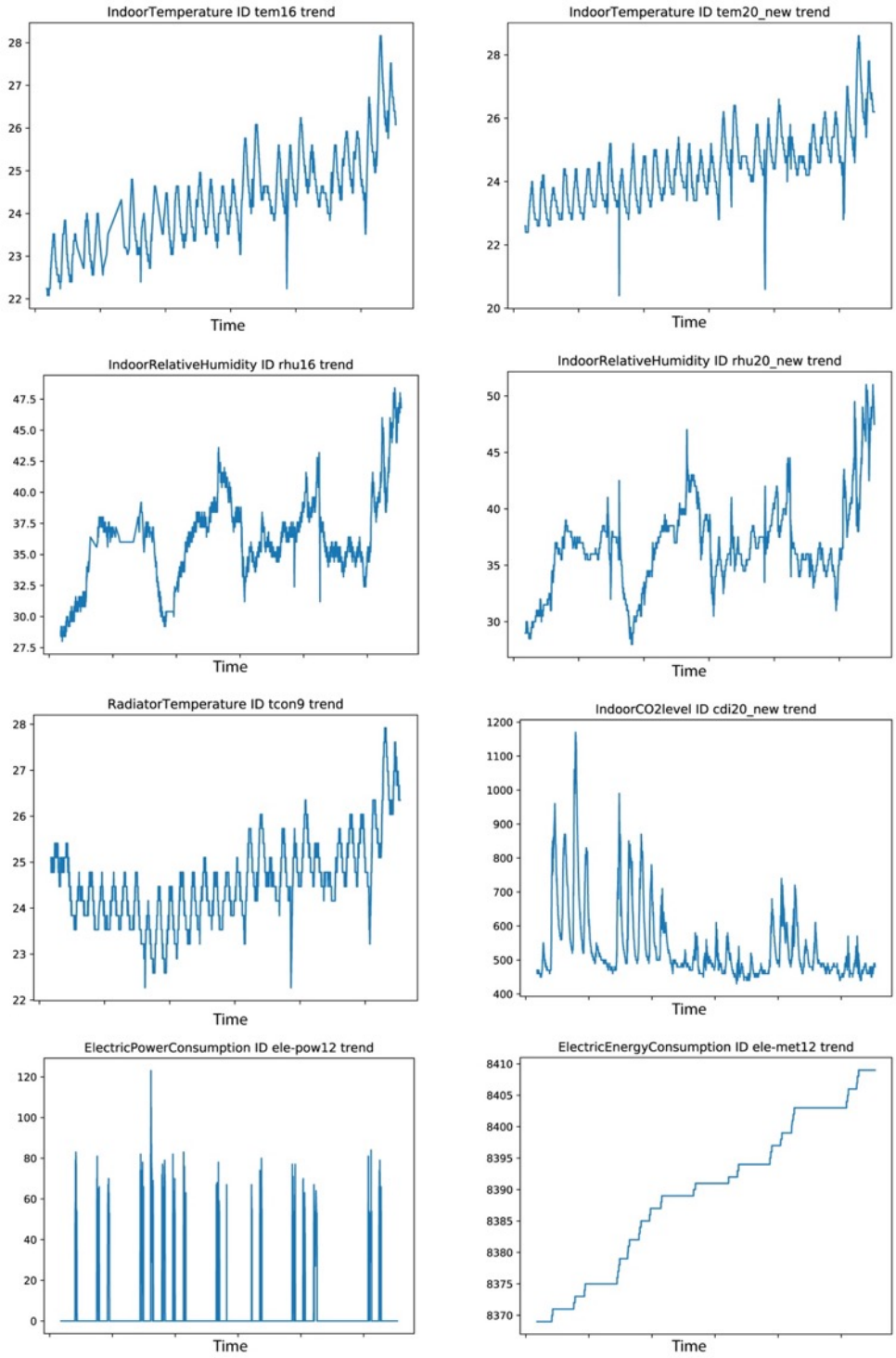

Figure 18 Trend preview of filtered variables in the selected time range (1-31.05.2016).

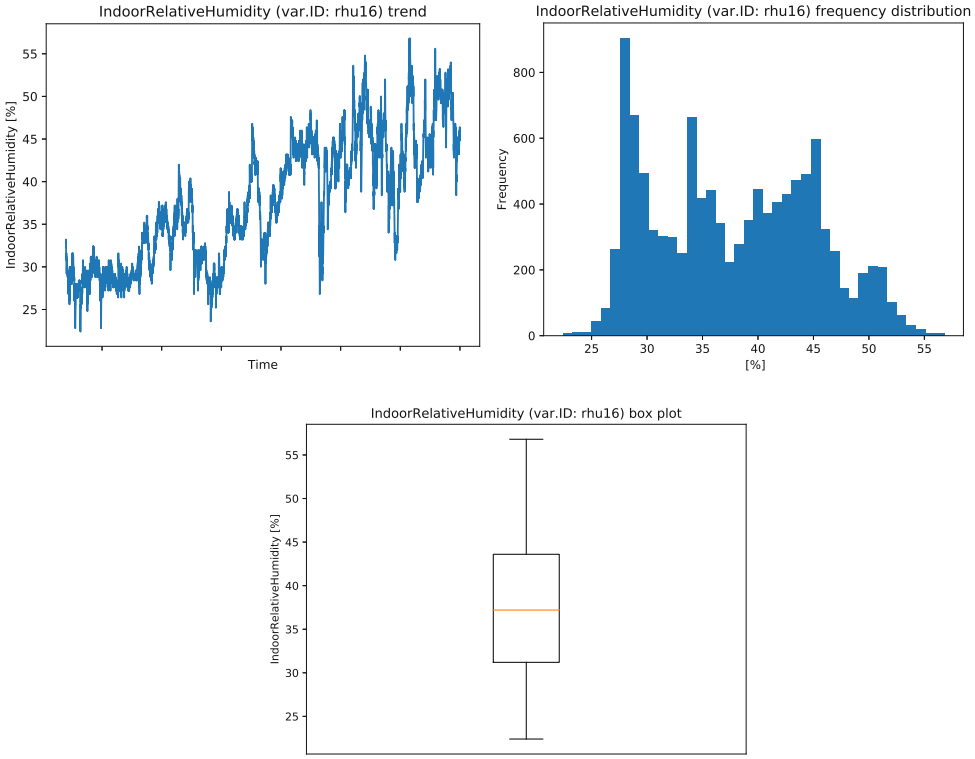

Figure 19 An example of statistically relevant visualizations, generated from query selected measured values of indoor air relative humidity variable instance (top: trend, frequency distribution; bottom: box plot).

The last "low-level" illustrative example, based on logical query information retrieval is indoor-outdoor temperature relationship study. It was hypothesized that, due to the effect of thermal mass and associated temporal delay, indoor air temperatures may display a higher correlation with outdoor temperature measured at an earlier instance. In this scenario logical query constrains were as follows: "indoor/outdoor air temperature" as variable names, "BPI open space S, TU tower" as topological reference, "21.06.2017 - 22.09.2017" as temporal constraint. Processing involved aggregation of available variable instances to a 30 minutes step temporal base and averaging the indoor air temperature values to get one representative indoor air temperature measure. The correlation between indoor air temperature with outdoor air temperature that was measured 30, 60, 90, 120, 150, and 180 minutes earlier was then investigated (see Figure 20). This analysis suggest that the highest correlation involves a two-hour time shift between measured indoor and outdoor temperatures. Figure 21 displays this correlation graphically.

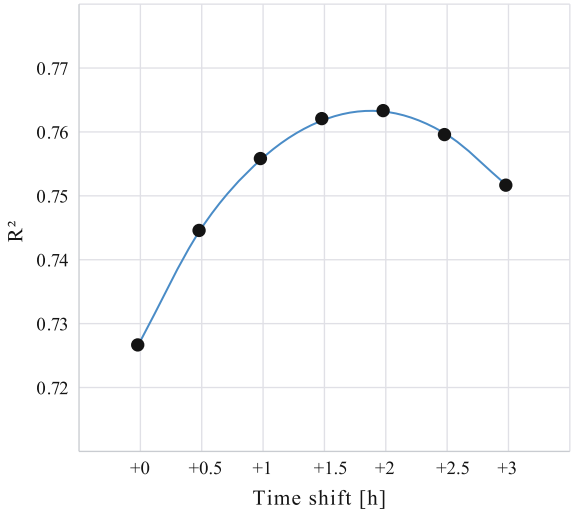

Figure 20 Coefficient of determination between measured indoor air temperatures in an office area and earlier measurements of outdoor temperature (from half an hour to three hours before).

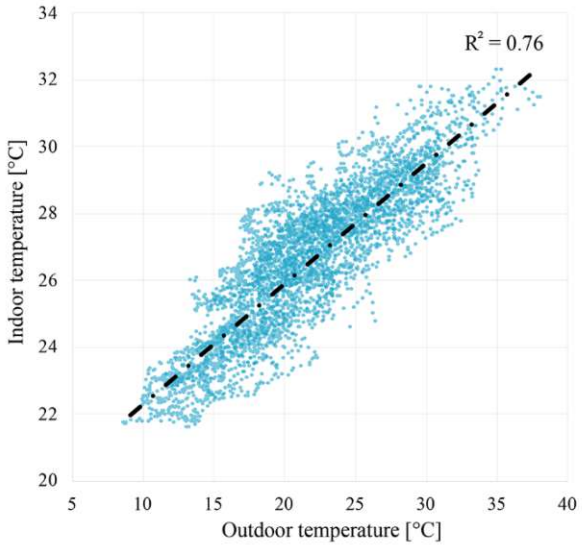

Figure 21 Illustration of the correlation between measured indoor and (two-hour shifted) outdoor temperatures.

3.2. Performance modelling tools integration

Note: The following section is based on and contains excerpts from the following, previously published conference papers - Wolosiuk and Mahdavi (2020a, 2020b).

To further explore the potential of utilization of semantically enriched data with advanced performance applications an effort was made to harmonize information stored in a HDF5 file and the functionality offered by the Ladybug Tools by creating task specific interfaces that relies on defined terms and data structure of BPD Ontology.

Ladybug Tools (LT) (Roudsari 2013) integrates the potential of well-known performance simulation engines such as the EnergyPlus (Crawley et al. 2001) or Radiance (Ward 1994), with the Rhino 3D (McNeel 2019) modelling software. It is a collection of small applications that couple these simulation engines with a 3D modelling and visualization potential of the Rhino software. This linking is enabled through Grasshopper – a visual programming environment built into the Rhino software.

Grasshopper quickly grew beyond initial 3D algorithmic modelling and parametric design platform. This was enabled by giving the community a possibility of creating custom components and component packages that could be shared online. The Ladybug Tools is an instance of such component package development. Grasshopper supports a number of programming languages for creating new components. These components can be of a universal nature (simple mathematical operations on input), as well as complex or task specific nature, such as calling external software for output generation. The latter is the case of this implementation, where several custom components written in Python language were created to enable and test interfacing between BPD ontology serialized in a HDF5 file and elements of Ladybug Tools.

Initial tests involved creating custom components that would take directly HDF5 file and some options as an input. The component's algorithm would extract a pre-specified performance variable in a desired quantity and process it to a form that is consistent with particular requirements of Ladybug's native component input.

One of the tested functionalities was to see if a sky matrix of radiation values representing amount of radiation coming from each of 145 patches of a sky hemisphere, could be used with different native LT components that require these values as an input. The values were generated from previously processed sky-luminance camera images (See Figure 22). For example, in this case, to be visualized as a 3d hemisphere in the graphical interface of the Rhinoceros 3D using the *SkyDome* LB component (see Figure 23).

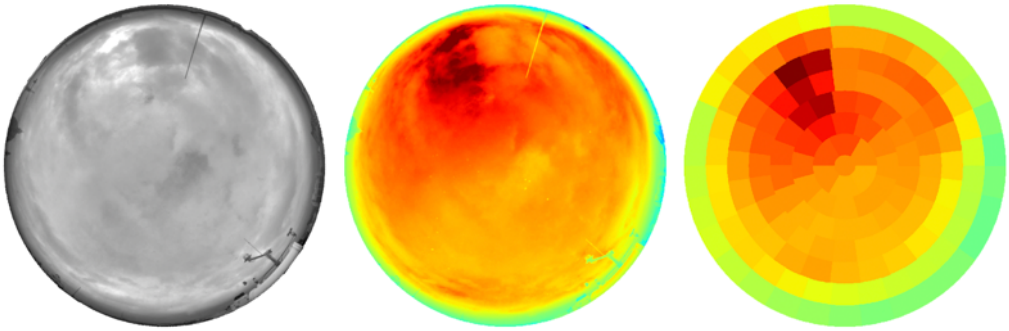

Figure 22 Sky luminance camera processing stages. From luminance image to Tragenza sky matrix.

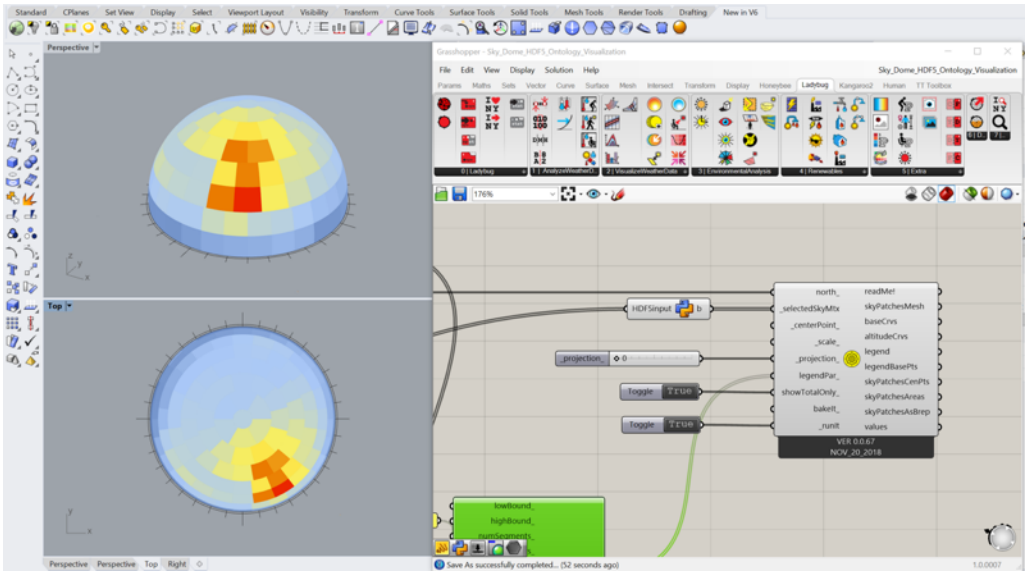

Figure 23 Sky matrix generated from a luminance camera image visualized in Rhinoceros 3D environment.

In another scenario, an application specific component was created to take measured radiation data from the HDF5 file and to replace Typical Meteorological Year (TMY) values in the original EPW file. Values recorded in a specified year (or a specific period within a year) are i) aggregated in terms of hourly values, ii) direct normal component is calculated from global and diffused horizontal radiation (if not provided), iii) latitude and longitude information is replaced (if provided), iv) and finally a new EPW file is generated, stored locally, and provided as an input for the Generate Climate Based Sky component. Figure 24 shows this component with the required and optional inputs on the left and outputs on the right side.

Figure 24 A custom component for modification of EnergyPlus weather file, for use with climate-based sky generator in solar radiation studies.

Being able to modify parts of an existing EPW allows for seamless integration of localized environmental data into the standard Ladybug design or analysis workflow.

The created BPD interfacing component was tested by integration in two illustrative simulations, involving snapshot-type Illuminance and annual Daylight Autonomy analyses. The test case for the created component involved modified sample studies provided by the creators of Ladybug software.

The illuminance simulation results are based on selected diffuse horizontal and direct normal irradiance values obtained from modified EPW file generated by a custom component. The new EPW file contains radiation values for the entire year as recorded by pyranometers in 2016 in Vienna city center. The selected point in time for the simulation is 2016-06-2110:00:00. Figure 25 presents the integrated BPD interfacing component into Ladybug components-based simulation setup (only a small part of the setup canvas is visible here). The resulting daylight illuminance distribution values are visualized in Figure 26 in terms of a color scale.

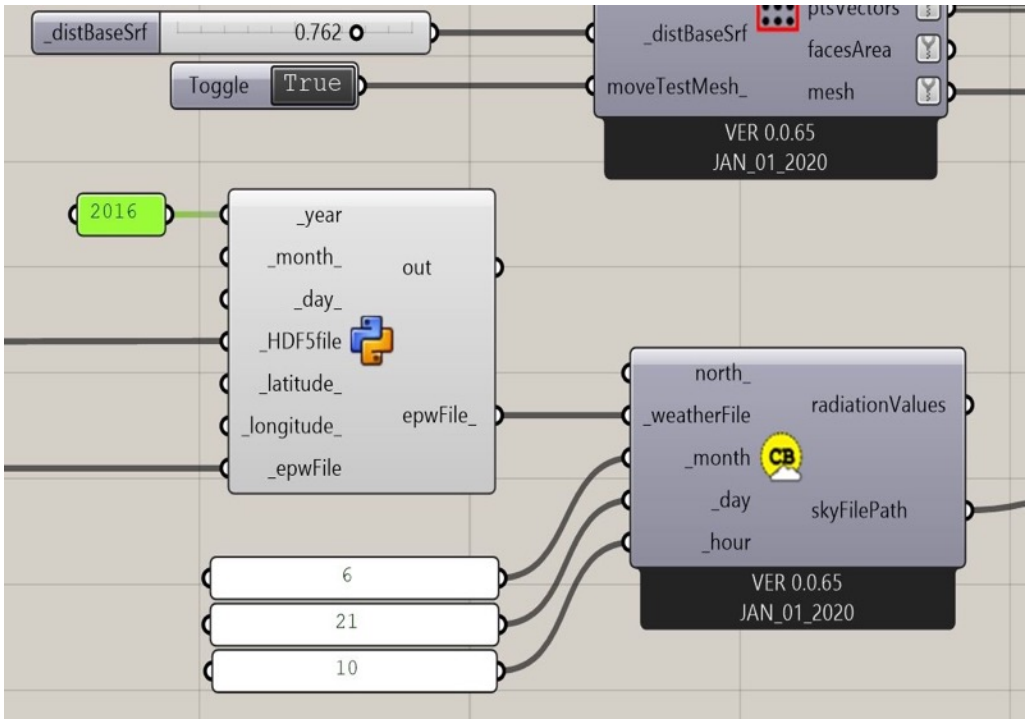

Figure 25 Custom BPD interfacing component (middle left) integrated into simulation setup.

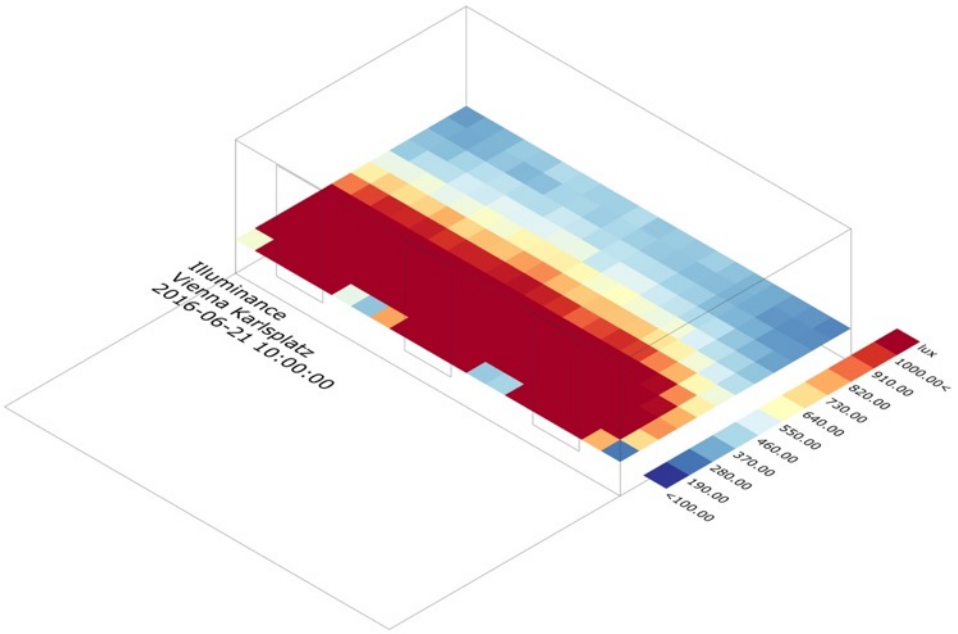

Figure 26 Visualization of the indoor illuminance simulation results based on local historical data for Vienna.

In a similar manner, an annual Daylight Autonomy (DA) for the same space was performed. This time the local historical solar radiation data from the entire year 2016 was used to visualize and analyze Daylight Autonomy based on a default office type occupancy schedule. The simulated results represent the percentage amount of occupancy time when the illuminance is above the given threshold of 300lux. Again, results are visualized as a color mapped grid representing value threshold for a given analysis grid tile (see Figure 27).

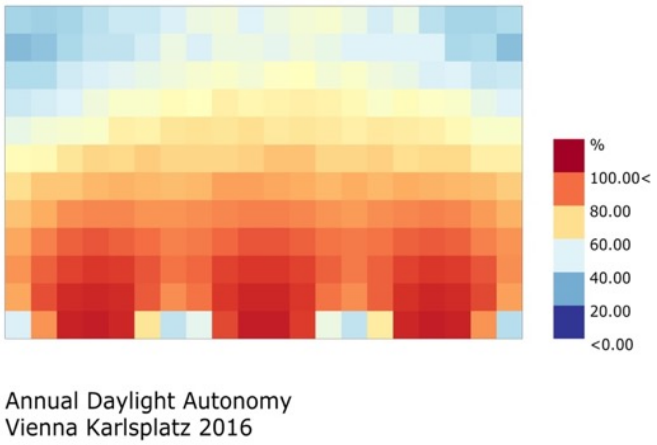

Figure 27 Daylight Autonomy studies based on local data extracted from BPD stored in a HDF5 repository.

After the possibility of extracting and harmonizing the ontologized BPD with Ladybug Tools via "hard-coded" selection of data had been demonstrated, in the next stage some usability related functions and components were added so that information stored in an HDF5 file can be accessed from Grasshopper's interface level. For example, a function for selecting variable instances based on attribute filtering via interactive interface elements (e.g. generated drop-down list) (see Figure 28). Another added functionality is data processing component, where values related to selected variable are extracted according to specified time frame and can be further aggregated and missing data can be interpolated (see Figure 29). These new components enabled interactive access to further group of Ladybug Tools components that use this input data streams for visualization, indices calculation, or performance simulation.

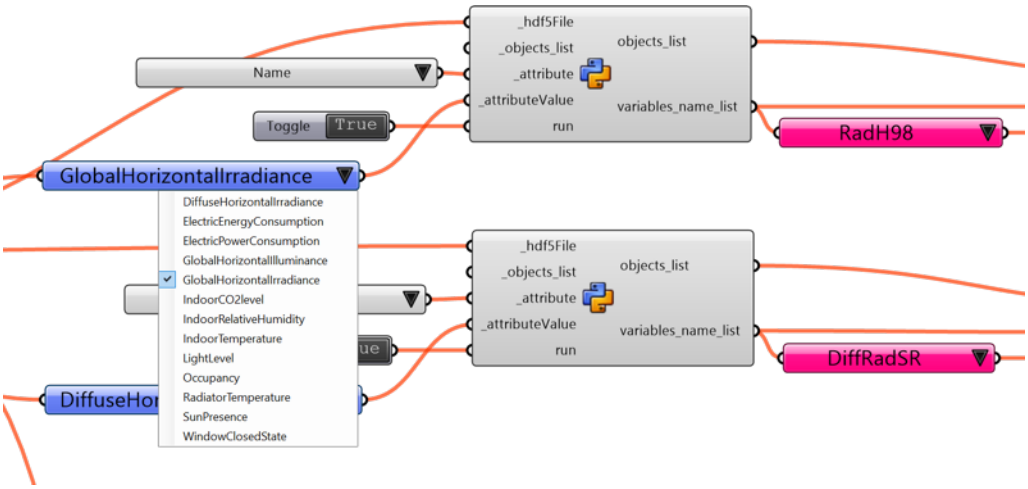

Figure 28 An illustration of custom components created for selecting relevant variable instances based on attribute filtering via interactive interface elements.

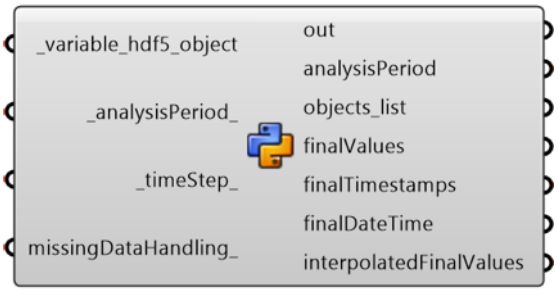

Figure 29 A custom data processing component for analysis period selection, data points aggregation or missing data handling.

For example, by selecting variable instances related to thermal comfort, such as the air temperature and relative humidity, regarding certain occupant in the selected office space, we could utilize the "PMV comfort calculator" or "Thermal comfort indices" LB components to deliver number of thermal comfort related indices.

There are several options for data visualization in Ladybug. In a basic visualization scenario, the selected temperature data points can be supplied to "line chart" component to generate a trend line graph. Figure 30 present a simple chart generated from one of the indoor air temperature variables ("tem20_new") in the specified time period.

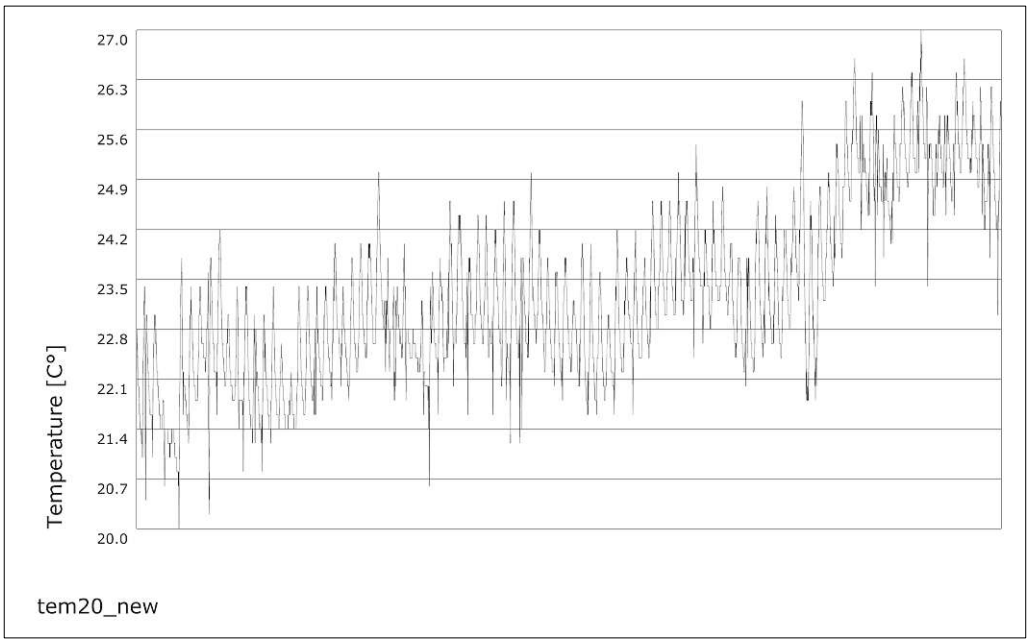

Figure 30 Example of a line chart generated with Ladybug native component from extracted indoor air temperature variable data.

Another visualization option is previously presented tile map. In this case native LB component is used to generate an hourly overview of a variable in question. Figure 31 presents a tile map of indoor relative humidity generated using "3DMap" component.

In an advanced visualization scenario, thermal comfort related variables are supplied to generate a psychometric chart. The typical process steps are as follows: First a topological reference attribute is selected from the list of available attribute instances in the HDF5 file (Figure 32 A), in the next step the relevant instances of the Indoor Relative Humidity and Indoor Air Temperature are selected (Figure 32 B), the analysis period is defined (Figure 32 C) and finally the event-based measured values are processed in terms of aggregation to

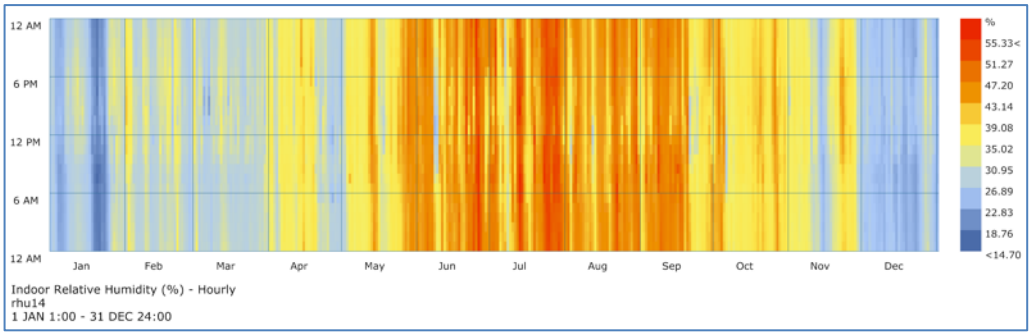

Figure 31 A tile map visualization of an indoor relative humidity variable, generated using native LB "3DMap" component.

a common hourly time-step (Figure 32 D). Finally, the data is supplied to the "Psychometric Chart" component. The resulting graph (see Figure 33) presents a color-coded density grid of total amount of hours in the specified time period of temperature – humidity pairs inside each of the grid tile (1C° and 5% Rh grid step). The total amount of hours within the comfort window can be read from the components output or the annual results can be visualized using "3DMap" component (see Figure 34).

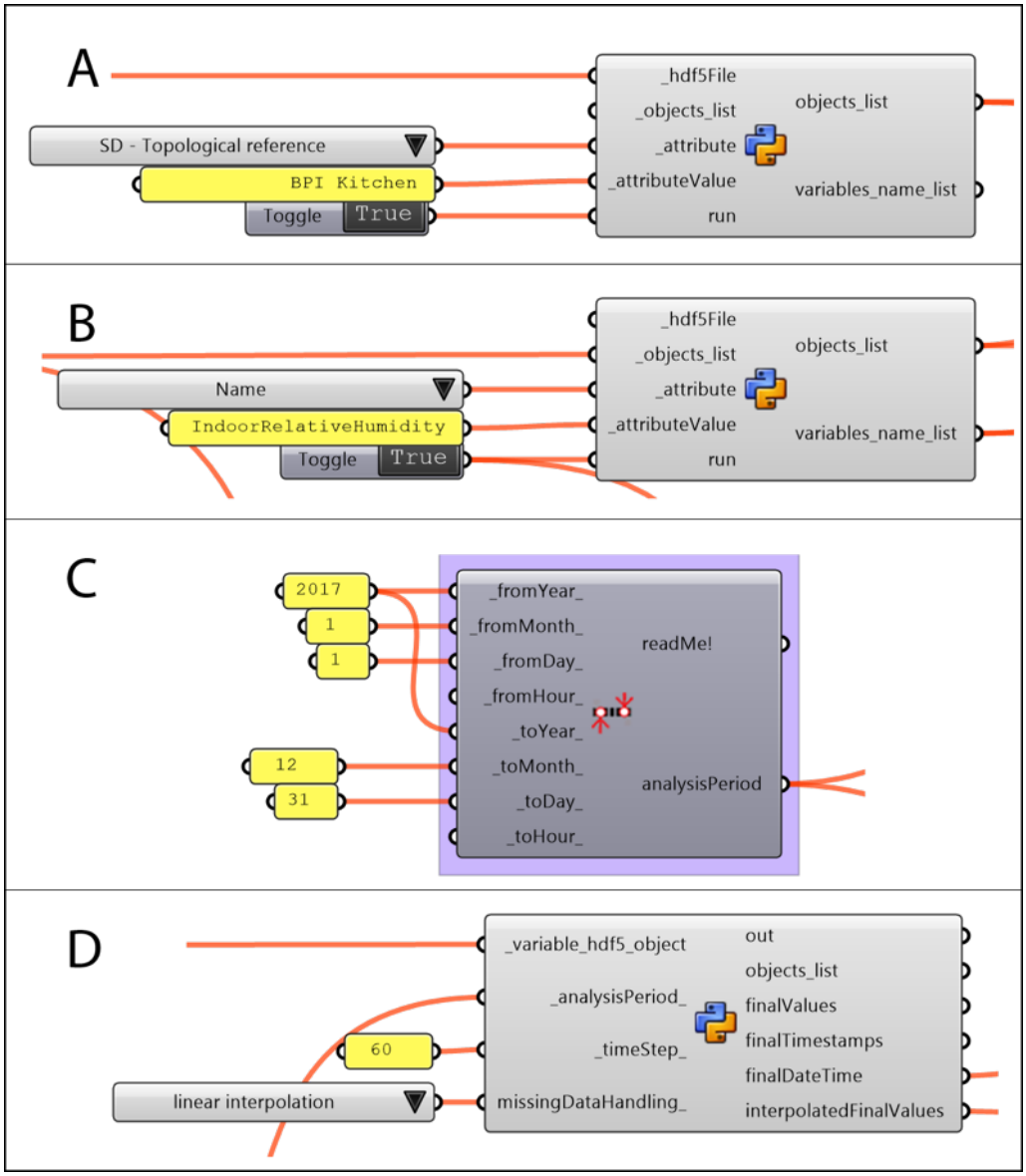

Figure 32 Stages in Grasshopper visual programing model for psychometric chart generation from ontologized data stored in hdf5.

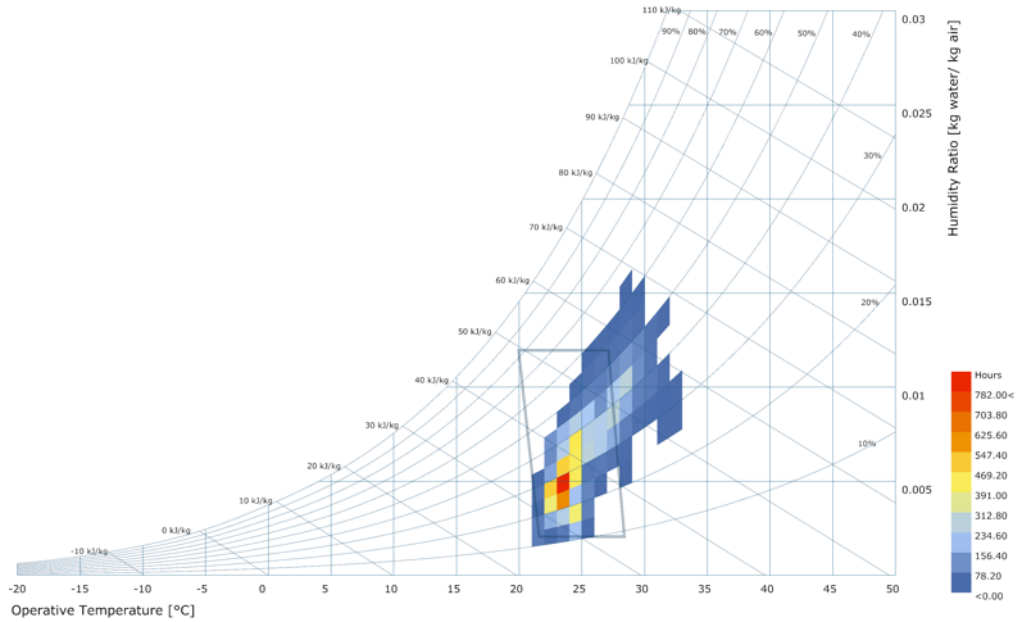

Figure 33 Psychometric chart generated from the performance data stored in hdf5 file.

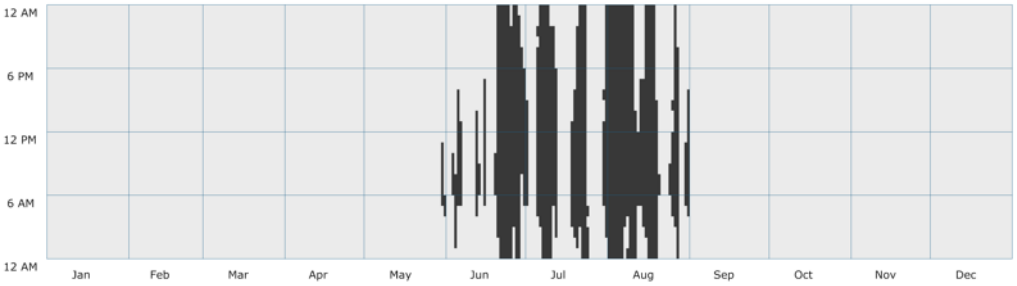

Figure 34 Annual hourly tile map showing hours of the year outside the comfort window.

In the last tested interfacing instance, the aim was to use detailed sky radiation data collected by a sky-scanner and direct normal solar radiation data (derived from global and diffused solar irradiance measurements) toward localized solar radiation studies. Sky-scanner regularly measures, for a specific location, luminance and radiance of the sky hemisphere, represented in terms of in a 145 segments or patches. A custom component was created to extract ontologically structured solar radiation data from BPD and process it in terms of the input format of the Ladybug's Sky Dome component for data visualization. The *selectedSkyMtx* output is an array of aggregated total, direct normal, and diffused radiation values per sky patch for a selected time period. Figure 35 presents a visualization of the annual diffuse sky radiation based on the data collected by the sky-scanner in 2016.

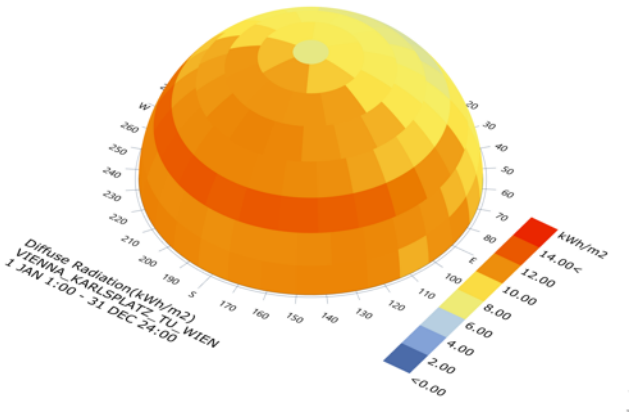

Figure 35 Visualization of diffuse solar radiation generated from sky-scanner measurements in Vienna, Austria.

The same structured (*selectedSkyMtx*) output can be used as the input for the Ladybug's Radiation Analysis component. It calculates the total incident solar radiation on a surface or a selected group of surfaces within a selected time period. It also allows for visualizing the results directly on the 3D model. Figure 36 presents the result of such a study of an existing building in Vienna. The Radiation Analysis results can be used in solar heat gain studies or support solar energy systems design.

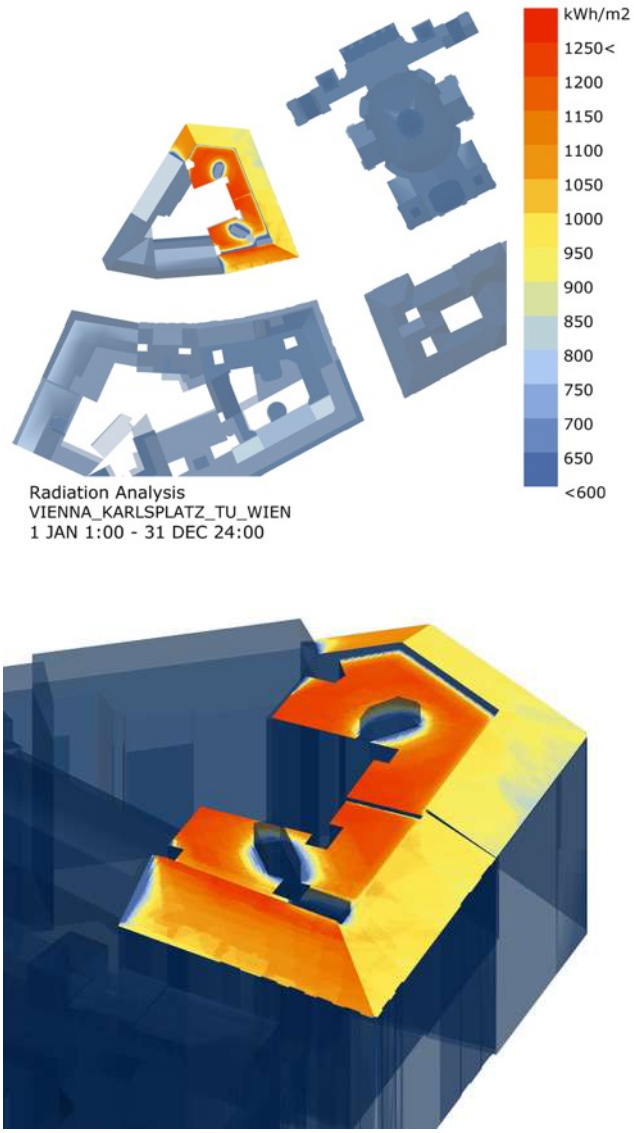

Figure 36 Illustration of a building performance assessment scenario (visualization of incident solar radiation density distribution across a complex roof configuration of an existing building) supported by ontologically stream-lined monitoring-based data.

3.3. Multi-domain PV performance studies

Note: The first example presented in this section is based on and contains excerpts from Mahdavi and Wolosiuk (2021). The remaining two examples are largely based on to be published conference papers – Wolosiuk et al. (2021) and Mahdavi at al. (2021).

Previous sections demonstrated how certain streams of data could be ontologized and then utilized in different application scenarios. There are, of course, many more cases of technical applications of models and assessments of performance analysis that would require a much larger number of data streams and sources. To illustrate such cases a series of use case scenarios that involved climate, building, building system, user, finance was created and tested. Specifically, three advanced examples concerning utilization of solar energy, that focus on PV system energy generation potential, and payback time considering various constrains related to PV system setup, building energy performance, related soft costs and other financial aspects. This means, an extensive scope of information in four categories is required to create models and support relevant calculations. This includes:

- Microclimatic data: Location specific solar radiation data, as well as air temperature, humidity, wind speed, precipitation to enable accurate and high-resolution, generated energy profile calculation.
- Building related information: In order to determine the correct size, placement and orientation of a building-integrated PV system, detailed information about the building is required, including its shape and geometry, structure and construction. Essential is also high-resolution data on the electrical energy demand profile of the building.
- Technical specification of the components of the PV system: Information regarding the type and performance of inverters, panels, batteries etc. is required for calculation of generated energy profile
- Financial and regulatory information: As with other types of systems, economically relevant data (design, installation and maintenance costs of the PV system as well as electricity purchasing costs and electricity export revenues) are a key factor in calculation of the revenue time and support of decision-making.

In these examples a certain amount of data could have been theoretically covered by the proposed ontology (e.g. climatic data, energy consumption profiles, occupancy data etc.) but not all (e.g. financial data, building geometry, construction details, equipment specifications etc.).

Gathering reliable data necessary for these use cases was a relatively cumbersome task. There was very few accessible information sources and input data had to be extracted from different industry reports, websites, databases or trade journals, then interpreted, verified and manually prepared for implementation. Each of these homogeneous sources without

a reliable metadata layer add a new complexity level that hinders the development of potentially useful performance assessment related applications.

In the following, the said three use cases demonstrate how the broad spectrum of information is used in combination with performance data streams to provide meaningful analyses. First example presents a web-based application created for a PV system design decision support. Second example looks at cost-benefit analysis of different technical PV system installation and maintenance options under varying location specific environmental conditions. Third use case presents a framework that facilitates a bi-directional approach to supporting the design and configuration of PV installations. These examples represent also future challenges for the research described in this work. Particularly those concerning creation of a fully seamless data environments in which multi domain information is accessible thus enabling further progress and discovery.

Introductory remarks on PV performance modelling use cases

The source of all simulation methods and models related to solar power calculation in all of the following use cases is *pvlib-python* library. The *pvlib-python* library, is a package of functions and classes, written in Python programming language, for simulating the performance of photovoltaic energy systems (Holmgren et al., 2018). The library implements multiple models and methods developed by PV Performance Modeling Collaborative group (PVPMC) for Sandia National Laboratories (Stein, 2012).

The applied modeling procedure requires inputs such as sun positions, solar radiation data, weather data (climatic data), solar array orientation, and solar equipment specifications. These are then used as an input for the provided functions and models. The geolocation metadata (latitude, longitude, and altitude), hourly values of solar radiation (direct normal, global horizontal, and diffuse horizontal irradiance) as well as air temperature and wind speed were extracted from a corresponding EPW data files (Crawley et al., 1999). The weather data provided by the EPW file is derived from Typical Meteorological Year 3 (TMY3) (Wilcox and Marion, 2008) and International Weather for Energy Calculations (IWEC) data sets (ASHRAE, 2001). It is used to determine solar positions, optimal tilts, in-plane irradiance and finally energy output for each of the locations and scenarios. The NREL Solar Position Algorithm (SPA) (Reda and Andreas, 2004) is used to determine solar positions on an hourly basis for the specified sites. The solar positions, radiation intensities, and panel orientations data is used in a transposition model function to estimate the total, normal, and diffuse in-plane irradiance (Hay and Davies, 1980). What fraction of this irradiance is in the end converted to electric current depends on weather conditions and the PV systems' efficacy. Functions implementing Sandia Array Performance Model (King et al., 2004) are used to calculate cell temperature and finally total amount of power generated by the system.

Any deviation from this modeling approach, as well as financial modeling aspects and PV systems technological details, can vary between cases and are highlighted in the example

descriptions.

A web-based tool for PV installation design decision support

In the first use case, the aforementioned web-based computational platform for supporting the design and optimization of PV installations was developed. To perform different types analysis scenarios, it must be supplied with data from all of the previously mentioned categories. The efficiency of the process by which such heterogeneous data sources are brought together can benefit significantly from the data ontologization approach described previously. Although not identical in all cases, the ontologization process for the different data streams is basically similar. Therefore, the focus below is on demonstrating the usefulness of the computational environment in responding to a number of illustrative use case scenarios.

The application was developed in the Python programming environment, using the Dash framework for building web applications (Plotly 2015). It was used to create a plain user-interface for input specification as well as output results visualization and inspection. Figure 37 presents an overview of the developed interface. The main visualization type for different analysis results presentation is a contour plot as seen on the Figure 38. Depending on the user's options selection - solar radiation on the PV panels, generated AC power by the PV panels, annual financial balance (difference between system investments and energy cost savings over a pre-defined period of time) can be visualized. The result values are plotted as a function of the panels' orientation (tilt and azimuth). The second implemented visualization is a line graph that shows the cumulative (annual) cost balance as a function of the number of PV modules and the module orientation for a specific building, a specific module / inverter type and a specific electricity price scheme.

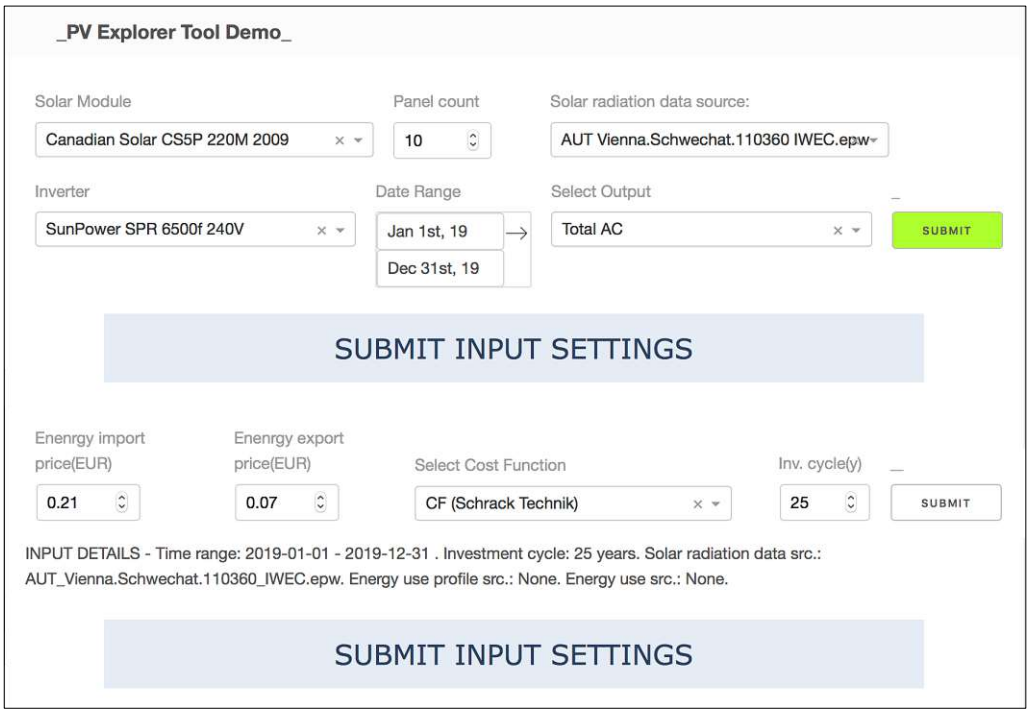

Figure 37 An overview of the developed tool's user interface.

In order to use the application, the user must provide some basic input data, including the applicable time period for the analysis, a source of local solar radiation (and other relevant microclimatic) information, and information on PV module products, and related inverter products. Depending on the type of output request, additional information regarding the number of modules and the time horizon for the financial analysis may be required. In addition, all inquiries about the local storage of electricity or electricity import / export from / to the grid require the electrical energy load profile of the building in which the PV system is housed.

To demonstrate the application and its functionality three illustrative use case scenarios were performed. In the first scenario user wants to analyze how much electrical energy can be generated by 15 PV panels over a period of a year under typical meteorological year conditions in a specific location (Vienna, Austria), given specific PV panel and inverter models (see Figure 37). The resulting visualization presents the respective values as a function of the azimuth and the inclination of the panel. Figure 6 shows the generated output, in the form of isolines of accumulated electrical energy given in kWh. While the calculated most optimal orientation is at an azimuth and an inclination of 165 and 35 degrees respectively, the visualization also illustrates the potential flexibility in a number of orientations that would provide a similar level of performance.

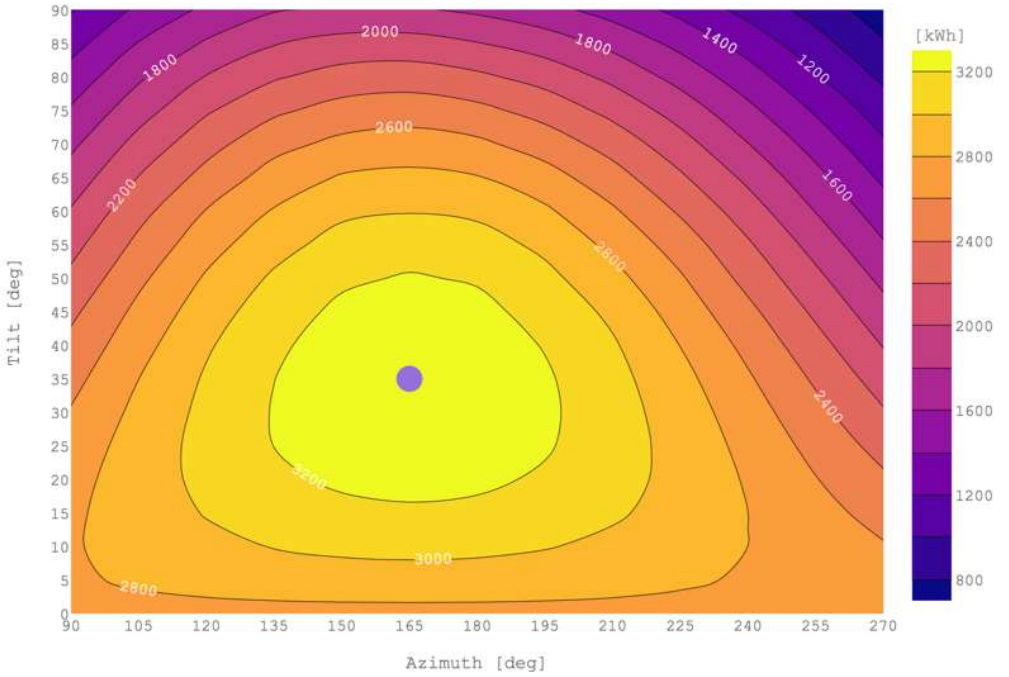

Figure 38 Isolines depicting computed cumulative electrical energy generated over a one-year period (15 PV panels installed in Vienna, Austria) as a function of the panels' orientation (azimuth and tilt)

The second illustrative use case scenario concerns the annual cost balance analysis of a building-integrated PV system consisting of 15 specific PV modules under the following input assumptions. The analysis is to be carried out for an investment cycle of 25 years. The total investment cycle costs consist of the purchase, installation and maintenance cost of the installation. It is assumed that if the electricity demand of the building (for the corresponding demand profile, see Figure 7) is not covered by the PV system, electricity must be drawn from the grid (price 20 cents per kWh). If, on the other hand, the system produces more electricity than required, the surplus can be fed into the grid (yield of 7 cents per kWh of exported energy). The final annual balance calculation is derived from the expenses for energy import, savings from the demand covered directly from the PV system as well as the income from the feed-in of PV electricity into the grid. The results of this example are shown in Figure 8 in the form of isolines of the annual cost balance (in euros). The calculated value is the difference between the total costs of the installation and the energy purchase costs over a period of 25 years and the income from the export of electricity into the grid over the same period.

Note that given the illustrative nature of this (and the following) application use case, the cost balance calculation has the character of a simple amortization analysis. In particular, the dynamics of energy prices and capital interests are not considered. However, these parameters can easily be included in the financial analysis component of the calculation method.

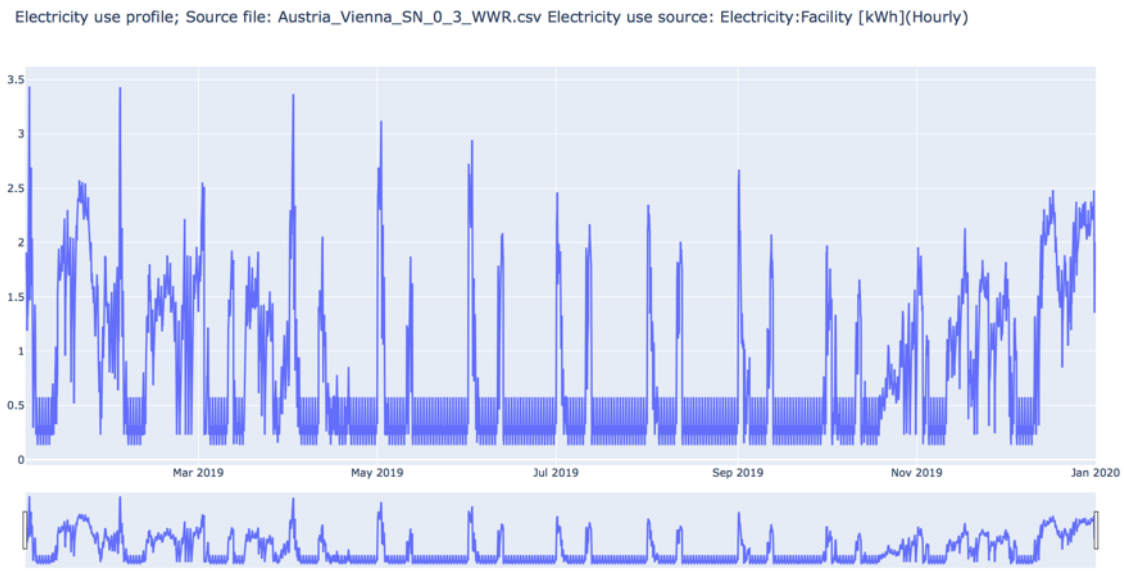

Figure 39 Assumed electricity use profile (hourly values over the course of one year)

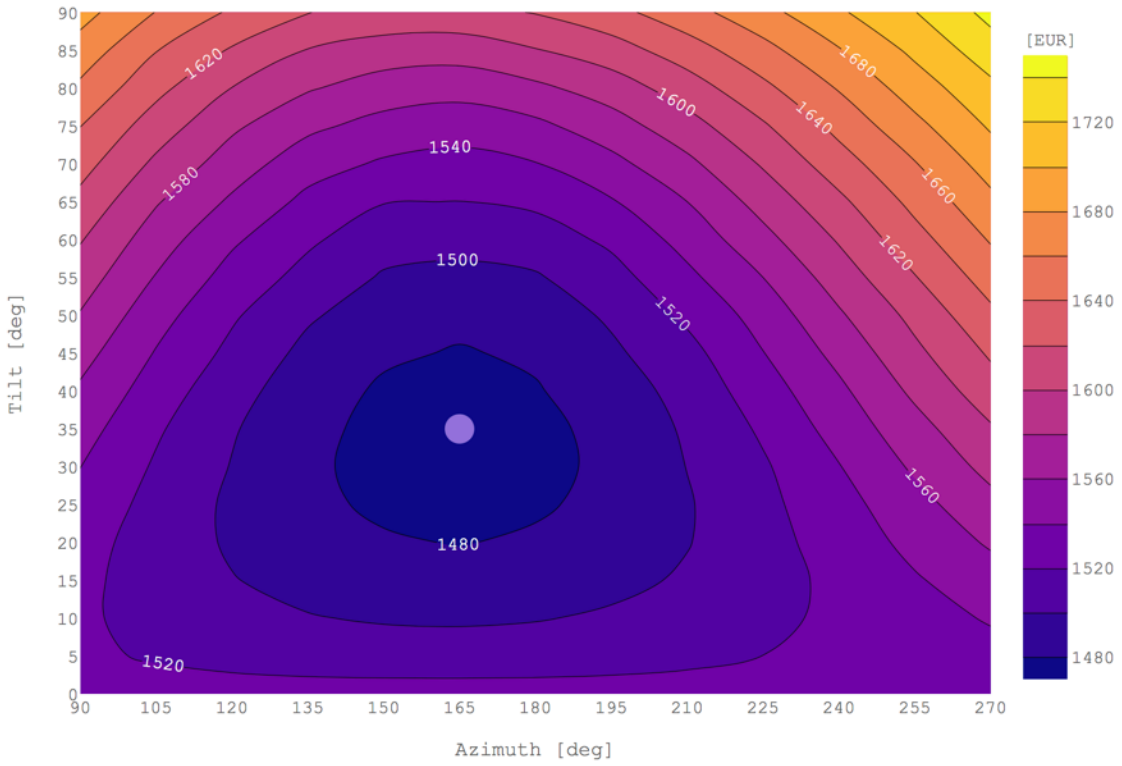

Figure 40 Isolines showing the calculated annual financial energy balance of a building with integrated PV system.

The third illustrative scenario tries to determine the optimal number of PV modules (of a certain type hence specifications) for an installation in the same location (Vienna, Austria). As in the previous case, the aim of the optimization is to minimize the annual net system costs (sum of installation and electricity import costs minus income from electricity export into the grid, calculated over a period of 25 years).

The application's interface allows selection of different orientations of the panels as well as different assumptions about the electricity import and export prices. To illustrate the results of the calculations, Figure 41 focuses on just one panel orientation (i.e. azimuth of 180 degrees and inclination of 35 degrees) and a fixed electricity purchase price, set at 20 cents per kWh. This figure suggests that up to a minimum electricity export price (i.e. around 8.5 cents per kWh), any increase in the number of modules leads to lower annual costs. Below this price threshold, however, a certain number of modules turns out to be optimal (e.g. 20 and 10 modules for 8 and 7 cents per kWh export price for electricity).

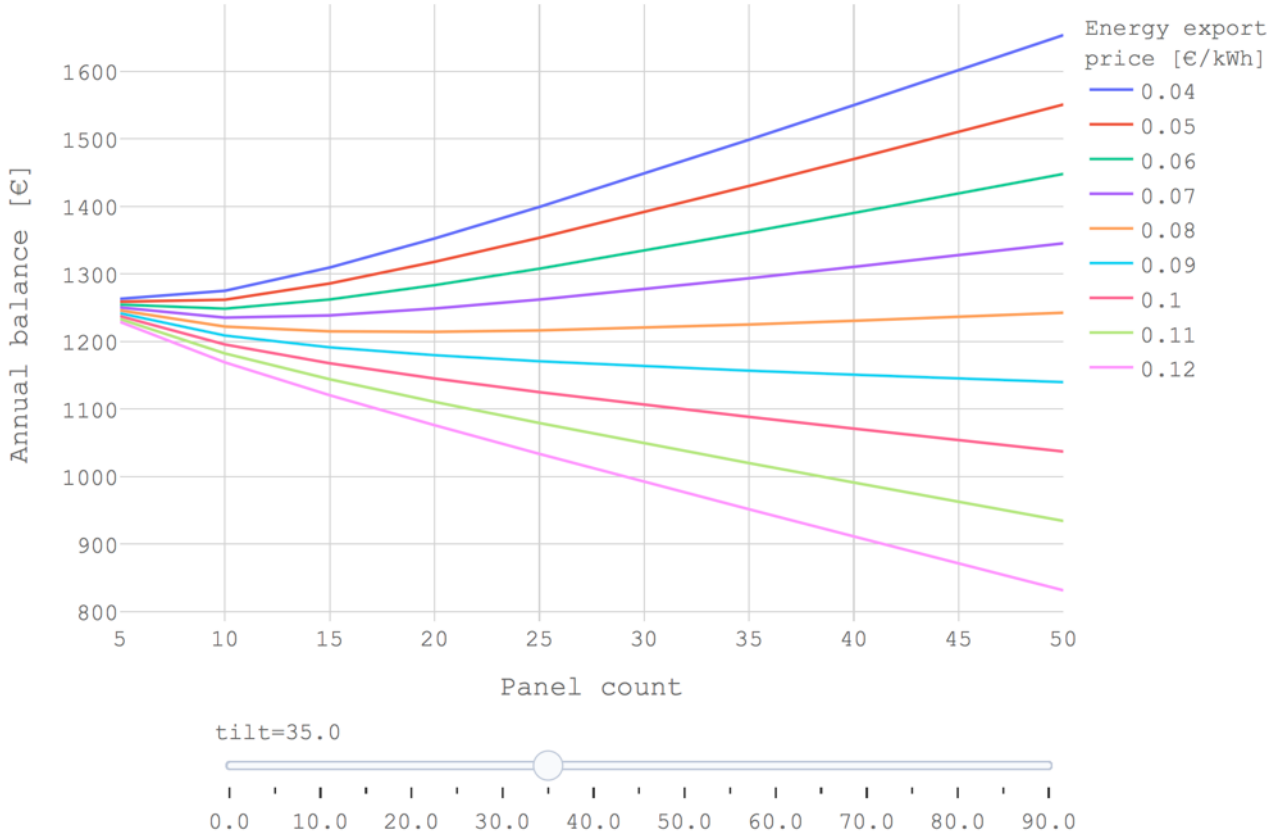

Figure 41 Calculated annual balance depending on the number of PV modules and electricity export price of a building-integrated PV system (panels orientation azimuth/tilt - 180/35 degrees, electricity purchase price set at 20 cents per kWh)

Performance comparison of static and adjustable photovoltaic panels

In the second example, the performance data streams are used to enable the cost-benefit analysis of static (fixed) installations versus dynamic installations capable of solar tracking. Moreover, in addition to entirely static option and the solar tracking options, it is conceivable to set up PV installations whose tilt could be "manually" adjusted from time to time to enhance their potential to capture solar energy. Whereas this latter option would be more expensive than conventional static installations (due to the needed mechanical gear and manual labor), it would require significantly fewer resources that automated solar tracking variants.

This example applies a high-resolution modelling approach together with order of magnitude cost-estimation to compare the energetic output and estimated installation and maintenance cost of static, fully dynamic, and multiple instances of recurrent manually executed directional adjustment of the PV panels.

Utilizing modeling methods and algorithms mentioned in the previous example, different panel slopes, directions could be parametrically assessed for the magnitude of generated electricity as a function of the solar radiation intensity at the installation's location.

The study scenario considered flat roof PV panels installation, where all panels were assumed to be of the same type, namely monocrystalline silicon (mono-Si) with a nominal module efficiency of 17.8% connected to a string inverter with a maximum efficiency of 97.5%. To streamline option comparison, obstruction issues (due to objects in the surroundings or due to adjacent panels) were not taken into consideration.

Four different locations were chosen for the study: Helsinki (Finland), Vienna (Austria), Santa Fe (USA), and Singapore. Table 4 contains basic geographic and climatic information on the locations for PV module installations. The source of the information included in this table regarding mean ambient air temperature, annual solar radiation, and annual sunshine hours are respective Energy Plus Weather (EPW) files.

Five PV installation options were considered, as per description in Table 5. F-xx denotes a static installation. Thereby, xx stands for the assumed panel tilt (degrees above horizon) for each location (40, 30, 35, and zero degrees for Helsinki, Vienna, Santa Fe, and Singapore respectively). The tilts were selected in such way to maximize annual solar gains given location specific input weather data. Options S-1, M-1, and M-2 denote manual adjustment of the tilt once every season, once every month, and twice every month, respectively. In all these cases panels are assumed to face South. E-W option denotes automated East-West solar tracking.

For the selected options and locations, the magnitudes of incident solar radiation and generated electricity were computed.

Table 4 Geographic and climatic information on the selected locations.

	Helsinki	**Vienna**	**Santa Fe**	**Singapore**
Latitude [°]	60.32	48.12	35.62	1.37
Longitude [°]	24.97	16.57	-106.08	103.98
Altitude [m]	56	190	1934	16
Mean annual air temperature [°C]	5.2	10.0	10.8	27.5
Annual solar radiation [kWh/m^2]	947	1122	1986	1671
Annual sunshine hours count	1669	1680	3554	1651

Table 5 Specification of PV panel options.

Configuration	**Remark**
F-xx	Static (fix) panel; tilt (xx) determined as a function of location's latitude
S-1	Panels' tilt adjusted once every season
M-1	Panels' tilt adjusted once every month
M-2	Panels' tilt adjusted twice every month
E-W	Automated sun tracking (East-West)

The optimal panel tilts for the manual adjustment scenarios were calculated as mean altitude of the sun at solar noon in the specific location per adjustment frequency period. In the more complex sun tracking scenario, the PV panel tilt is constantly adjusted to follow sun on its path from east to west. An algorithm that minimizes the solar beam angle of incidence and translates it to a so called true-tracking rotation angle is applied to determine a panel tilt angle that maximizes solar gains (Marion and Dobos, 2013; Anderson and Mikofski, 2020). The solar positions, radiation intensities, and panel orientations data were used to calculate total amount of power generated by the system in each of the cases.

In addition to computed energy magnitude, an effort was made to address the financial ramifications of these options. To this end, solar equipment and installation costs, as well as maintenance costs were estimated. Note that given multiple sources of significant uncertainty in all relevant input assumptions for this calculation, this calculation is not intended to be a final and accurate assessment. Such uncertainties pertain to relevant cost items (installation, maintenance, labor etc.) and energy tariffs as well as their future development. Rather, the purpose of this exercise was to provide a preliminary order of magnitude impression of the financially relevant aspects of the subject. It is thus also

important to explicitly mention some of the major simplifications made in the calculation process. For instance, no differentiation was made between the different locations with regard to the assumed unit costs for first installation and maintenance the different PV configurations. Likewise, no location-based differentiation was also made with regard to the electricity price and its dependence on the fraction of the energy used by the building versus the fraction exported to the grid. To establish a pertinent electricity pricing range for option comparison, two contrasting positions at the two ends of use scenarios for the generated electricity were considered. Whereas at one position it was assumed that all generated electricity is locally used, at the opposing position it was assumed that all electricity is supplied to the grid.

Estimated cost of a fixed PV installation is based on market price in Austria (Biermeyr et al., 2020). The cost of manually adjustable system was assumed to be 1.5% higher than that of a fixed system. This assumption is based on the market price difference between fix and adjustable mounting systems, as well as the fraction of the racking system in relation to the overall installation cost. The cost of the one axis tracker PV installation was assumed to be 10% higher than that of fix version (Fu et al., 2018).

The annual operation and maintenance costs were assumed to amount to 1 to 1.5% of the investment costs for residential PV system (Fu et al., 2018). Following the pattern of installation cost difference between fix and tracking systems, the base maintenance cost for the tracking system installation was assumed to be 10% higher than in other scenarios.

Additional labor costs related to manual adjustment of the position of the panels depend on the required frequency of adjustment and related manual labor cost. The estimated hourly labor cost in the services sector in Austria is currently about 32 euros (Eurostat, 2020a).

The estimated annual monetary gains are based on average electricity prices for household consumers (0.21 euros.kWh-1 according to Eurostat 2020b) and electricity feed-in tariff (0.077 euros.kWh-1) according to RIS (2017) in Austria.

The resulting graphs as seen on Figure 42 to Figure 45 display the computed monthly quantities of generated electricity per unit PV panel area for the aforementioned five installation options and four installation locations.

Table 6 includes computed annual values of generated electricity (in kWh.m-2) for the five PV configuration options and the four locations. Table 7 entails the same information in relative terms, that is the percentage deviation of the options with reference to the fixed tilt (F-xx) option.

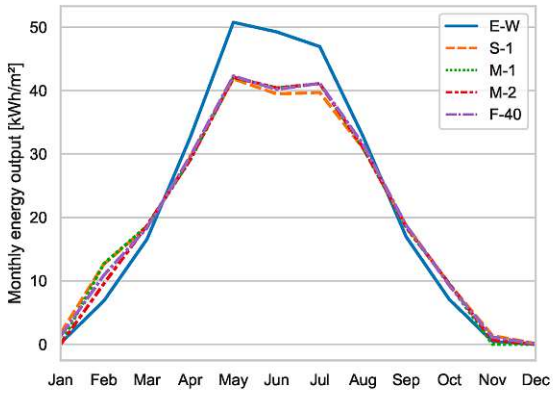

Figure 42 Monthly values of generated electricity (in kWh.m-2) for the for the location Helsinki.

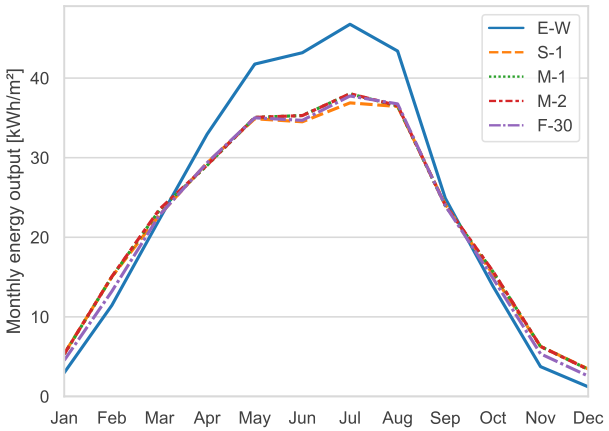

Figure 43 Monthly values of generated electricity (in kWh.m-2) for the for the location Vienna.

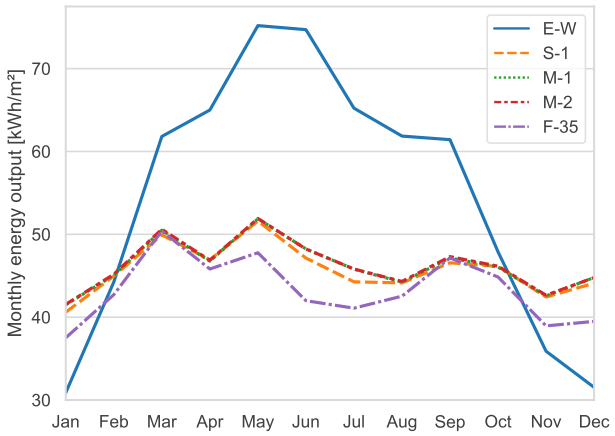

Figure 44 Monthly values of generated electricity (in kWh.m-2) for the for the location Santa Fe.

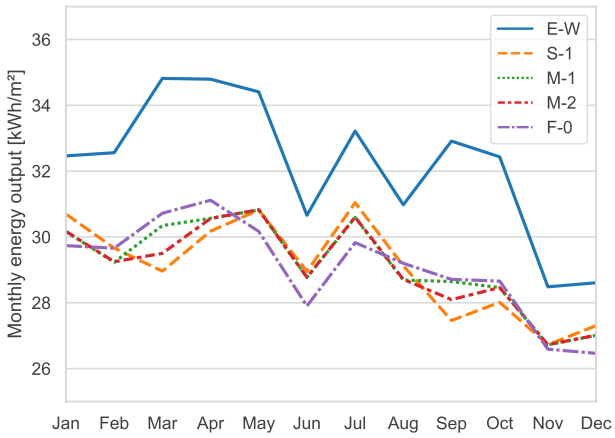

Figure 45 Monthly values of generated electricity (in kWh.m-2) for the for the location Singapore.

Table 6 Annual values of generated electricity (in kWh.m-2) for the five PV options and the four locations.

PV Option	Helsinki	Vienna	Santa Fe	Singapore
F-xx	137	155	309	207
S-1	137	157	326	207
M-1	139	159	330	208
M-2	138	159	330	207
E-W	146	171	390	229

Table 7 Deviation (in %) of the annual values of generated electricity via options S-1, M-1, M-2, and E-W from the fixed option (F-xx).

PV Option	Helsinki	Vienna	Santa Fe	Singapore
S-1	0.0	1.3	5.5	0.0
M-1	1.5	2.6	6.8	0.5
M-2	0.7	2.6	6.8	0.0
E-W	**6.6**	**10.3**	**26.2**	**10.6**

As alluded to before, to compare the options in economic terms, no differentiation was made between the different locations with regard to the assumed unit costs for first installation and maintenance the different PV configurations. Likewise, no differentiation was made with regard to the electricity price and its dependence on the fraction of the energy used by the building versus the fraction exported to the grid. Specifically, the scenario involving full self-sufficiency assumed to correlate with 0.21 Euros.kWh-1 saving, in contrast to the full export-to-grid scenario, which was assumed to result in 0.077 Euros.kWh-1 gain.

Table 8 provides an overview of the results of the economic analysis. It includes the unit cost of installation (I), maintenance (M), annual electricity-based monetary gain for 100% self-sufficiency (GS) versus 100% export to grid (GE), payback times (PB in years) for additional costs as compared to the F-xx scenario for both electricity price schemes (PBS and PBE). Payback times are based on simple payback analysis.

Table 8 Economic comparison of the options, including assumed unit cost (in euros.m^{-2}) of installation (I), maintenance (M), annual electricity-based monetary gain for 100% self-sufficiency (G_S) versus 100% export to grid (G_E) as well as payback times (PB in years) for both electricity price schemes (PB_S and PB_E).

		F-xx	S-1	M-1	M-2	E-W
	I	295.7	300.2	300.2	300.2	325.3
	M	3.7	5.5	9.0	14.3	4.9
Helsinki	G_S	28.8	28.8	29.2	29.0	30.7
	G_E	10.5	10.5	10.7	10.6	11.2
	PB_S	10.4	10.6	10.6	10.8	10.8
	PB_E	28.5	29.1	29.0	29.7	29.5
Vienna	G_S	32.6	33.0	33.4	33.4	35.9
	G_E	11.9	12.0	12.2	12.2	13.1
	PB_S	9.2	9.3	9.3	9.4	9.2
	PB_E	25.2	25.4	25.4	25.8	25.2
Santa Fe	G_S	65.0	68.5	69.4	69.4	82.0
	G_E	23.7	25.0	25.3	25.3	29.9
	PB_S	4.6	4.5	4.5	4.5	4.0
	PB_E	12.6	12.2	12.2	12.4	11.0
Singapore	G_S	43.5	43.5	43.7	43.5	48.1
	G_E	15.9	15.9	16.0	15.9	17.6
	PB_S	6.9	7.0	7.1	7.2	6.9
	PB_E	18.9	19.3	19.4	19.8	18.8

Table 9 Deviation (in %) of the estimated payback times for options S-1, M-1, M-2, and E-W from the fixed option (F-xx).

PV Option	Helsinki	Vienna	Santa Fe	Singapore
S-1	2.1	0.8	-3.2	2.1
M-1	**1.8**	0.7	-3.3	2.8
M-2	4.3	2.4	-1.7	5.0
E-W	3.5	**0.0**	**-12.6**	**-0.3**

Table 9 facilitates the comparison of the estimated payback times in percentage terms, that is the percentage deviation of the options S-1, M-1, M-2, and E-W with reference to the fixed tilt (F-xx) option.

The comparison of PV installation options in view of their electricity generation potential (as shown in Figure 42 toFigure 45 and

Table 6 andTable 7) warrants certain conclusions. The added value of manual adjustment of panels' tilt is de facto negligible (less than 3%) in case of three locations (Helsinki, Vienna, Singapore) and rather modest in one case (Santa Fe, 5 to 7%). Specifically, the frequency of these adjustments is inconsequential for all practical purposes and even counterproductive (i.e., reduced gain by higher adjustment frequency) in some cases.

From purely energetic output point of view, as compared to the fixed installation option, the automated E-W tracking option does appear to yield notably higher magnitudes, that is from around 7% (Helsinki) to 26% (Santa Fe). However, the question remains if this increased output justifies the necessary additional investment and maintenance costs. To discuss this point, consider the data summarized in Table 8 andTable 9. This data appears to suggest that, with the exception of the E-W option, the estimated payback times do not significantly vary across the installation options considered. However, assumptions regarding energy tariffs influence the payback times of PV panel installations significantly. In the present case, moving from full internal utilization of generated electricity to full export to the grid results in an almost threefold increase of the payback times. In summary, the results imply that:

i. PV panel installations may be suggested to have reasonable payback times in general, independent of the installation option and location, especially if a good fraction of the generated electricity could be locally utilized.

ii. From the electricity production point of view, the manual adjustment options considered in the present study, do not appear to offer a noteworthy advantage as compared to the simpler fix-tilt option. The E-W option, on the other hand, does offer higher electricity yields, particularly in locations with higher solar radiation intensity.

iii. From the standpoint of payback time, only the E-W option at Santa Fe location displayed a significant reduction potential (approximately 13%). In the other cases, the advantage of slightly higher electricity yields via adjustable options was offset due to the necessary additional investment and maintenance costs.

As alluded to before, these results are to be regarded and interpreted cautiously. The aforementioned uncertainties – particularly with regard to financial factors (installation and maintenance costs) and the future evolution of electricity prices and tariffs – necessitate the careful reassessment of the observed trends within the actual (individual) context and circumstances of the settings considered for the installation of a PV system.

An exhaustive treatment of these uncertainties cannot be provided in the present contribution. However, in order to exemplify the influence of selected model input assumptions on option assessment results, a few basic instances were considered. One such instance pertains to a conceivable option, whereby, the manual PV panel adjustment, as well as basic maintenance (visual inspection, cleaning) are performed by building owners themselves, hence incurring no costs. The assessment of this scenario suggests that the

overall payback time in comparison with the base case would not be significantly reduced (ranging from no reduction at all to a maximum of 4%). Naturally, a more significant reduction of the payback time can be expected, if the installation of PV system is supported via subsidies. To demonstrate this, a scenario involving an installation subsidy at the level of 250 euros per kWp (City of Vienna, 2021) was probed. This level of subsidy results in an effective 16 to 18% reduction of the installation cost. This translates in turn, depending on the electricity pricing and location, into a relative payback time reduction of 2 to 17% as compared to the base case.

A bi-directional approach to building-integrated PV systems configuration

The configuration of local building-integrated photovoltaic (PV) installations can benefit from effective and reliable computational support. Especially in cases where a high degree of energy self-sufficiency is desired, it is important to optimally match the temporal profiles of the building's energy demand and the available solar radiation intensity. In many instances, such a matching is conducted in a mono-directional manner. As such, the building's demand profile is taken as given, which is treated as the basis for the sizing and configuration of the PV installation. The computational framework introduced in this research facilities this type of matching, but it is intended to offer additional functionalities. Specifically, the developed computational platform is conceived to facilitate a bi-directional approach to supporting the design and configuration of PV installations meant to be integrated in new building projects. Thereby, the idea is to probe pertinent building design variables such as orientation, transparent envelope elements, thermal mass, daylight use, and indoor climate control systems (for heating, cooling, ventilation, and lighting) in terms of the magnitude and temporal distribution of the resulting building energy demand. The proposed bi-directional iterative approach not only informs the configuration of PV system based on the building's demand profile, but also allows for the exploration of the consequences of the magnitude and temporal profile of the PV's energy supply potential for the aforementioned relevant building design variables. In other words, the user can move from the direction of a given building's energy demand profile toward derivation of appropriate PV installation attributes, or from the opposite direction of a given PV-based energy supply profile to explore implications for optimized building design variables. This example presents said computational approach and its functionality in terms of an illustrative case study.

The computational platform for the aforementioned bi-directional configuration support of building-integrated PV systems entails three components. These components, which are briefly described below, serve the i) parametric computation of a building's energy demand profile, ii) parametric computation of a PV installation's electrical energy generation potential, iii) computation, visualization, and navigation of the values of a number of whole-system performance indicators.

The computation of the building's energy demand (and its temporal profile) involves a

number of steps. In a first step, the geometry of the case study building is modelled in the 3D modelling environment Rhino. In a second step, the geometry model is augmented with required input assumptions by using the integrated visual scripting platform Grasshopper together with the Ladybug Tools plug-ins. Thereby, information about the building with regard to the location, construction, occupancy, equipment load, lighting density, and the heating, cooling, and ventilation system is specified. After the required input data is defined in Honeybee, the building energy simulation are conducted using EnergyPlus.

The computation of the PV-based electricity similar to the previous two examples made use of the Python programming language and *pvlib-python* library to calculate solar position, optimal tilts, incident irradiance, and generated electricity for each of the applicable location and scenario.

To estimate the amount of time it takes to recover the cost of a solar PV system investment, one needs to look at two numbers, namely the estimated total system's lifetime cost and the annual monetary gains from the energy generated by the system. The cost of the former consists of system's initial purchase and installation price and the lifetime maintenance cost. The estimated purchase cost depends on the solar array size and is based on a market price kW peak (kWp) power (in Austria ca. 1650 €.kWp-1; Biermayr, 2020). The annual maintenance costs are assumed to be at a level of 1% of the initial PV system's investment cost (Fu, 2018). Over the assumed lifetime of the system (25 years), the maintenance costs should cover the renewal of the inverters, as well as panel servicing and cleaning.

The annual financial gains depend on the energy usage profile of a building, energy generation profile of a PV system, and market electricity prices, both consumption and feed-in tariffs (roughly 0.21 €.kWh-1 and 0.077 €.kWh-1 in Austria; Eurostat 2020; RIS 2020). The annual gain is the sum of monetary value of the energy saved through the local coverage of the building's energy demand and the sale of the surplus energy. Within the framework of a simple payback analysis adopted in the present study, the final payback duration is the ratio of a total lifetime solar PV system cost to the annual monetary gains generated by the PV system.

To demonstrate the working of the proposed approach, consider the simple case of a building design project involving a set of eight identical row houses located in a site in the city of Vienna, Austria (see Figure 46). For the purpose of the present illustrative case study, certain attributes of these houses are assumed to be fixed, including overall shape and geometry, the U-value of external walls

(0.26 W.m-2.K-1) and windows (1.21 W.m-2.K-1), the occupancy density, and the heating, cooling, and ventilation system (electricity based). Other aspects were considered to be open for parametric analysis in conjunction with the PV installation options. Specifically, the façade glazing fraction could be varied (from 20% to 40%), the row of buildings could be rotated to face different orientations, and the ventilation rates could vary from a basic constant rate (0.4 h-1) to options involving summer-time ventilative cooling.

The technical specification of the solar PV system under consideration corresponds to a typical residential installation quality. The solar panels are considered to have nominal power of 300 W and an efficiency of 17.8%. They are to be connected inverter units with a maximum efficiency of 97.5%. Three aspects of the PV panels array were opened for parameterization, namely the system size (18, 36, 54 kWp), the inclination of the panels (15°, 30°, 45°) and the alignment of the panels (SE, S, SW).

As stated previously, the proposed platform allows for the concurrent parametric modelling of both building configurations and PV system configurations. To demonstrate the working of the platform, the illustrative case of a building design project involving eight row houses was considered. This building was to be equipped with a roof-top PV installation. Both building and installation could be subjected to parametric analyses.

Given the demonstrative nature of the present treatment, a rather reduced set of both the building and the PV systems was considered for parametric variation. These variables included, in the case of the building block, the orientation of the building, the fraction of glazing in the façade, and the ventilation rates. In case of the PV system, its overall size, as well as the tilt and azimuth (orientation) of the PV panels were considered for parametric analysis. Starting from a base case involving no PV system, it was possible to navigate through the design-performance space with the overall objective of reducing the payback time for the PV installation investment, or to increase the return on investment on such a PV system.

Note that, this payback time is focused on the PV installation only, and does not address the expenditures for the building itself. Certain changes in design variables (such as the glazing to wall ratio), however, may influence the building construction investment. Whereas the present case study did not consider the global payback time for both building and PV-system investments as the designated performance indicator, this can be implemented in the system, given the availability of pricing information concerning various building design options.

These simplifications notwithstanding, the prototypical implementation of the proposed approached displayed a promising capacity to support the navigation of option space and the convergence toward increasingly high-performance solutions. Note that the proposed navigation strategy is distinct form a one-time optimization approach meant to identify the optimal solution within the design-performance space. Rather, the idea is to support an open user-driven iterative and bi-directional search, thereby exploring different constellations of the building and PV-system attributes in view of their implications for the designated performance indicator.

To exemplify the characteristics of such navigation processes, Figure 47 illustrates a sequence of successive changes to either building or PV systems variables leading to steady reduction of the value of the selected performance indicator, that is, in this case, the payback time for the investment costs of the PV systems and their maintenance

expenditures. Note that, due to the difference in the electricity buying and selling prices, the magnitude of the performance indicator, namely payback time, is influenced by the level of matching between the temporal profiles of the building's electricity demand and the PV installation's supply. Obviously, given the assumed pricing scheme, variants that maximize the coverage of electricity demand via PV-based electricity are advantageous. Similarly, building design solutions that increase electricity demand during low-supply periods lead to the need for increased purchase of high-price electricity and are thus disadvantageous in view of the selected performance indicator. The data set generated through the aforementioned navigation process and depicted in Figure 47 is rather extensive. As such, Figure 47 shows the evolving – continuously improving – value of the performance indicator (payback time, y-axis) over the entire course of the convergence process (x-axis) in terms of the successive states of the designs.

For visualization and analysis purposes, this data can be disaggregated in different ways. For instance, Figure 48 shows the data in terms of three distinct sets, each corresponding to a different overall PV system size. As mentioned before, the various positions in these functions represent different design states (i.e., different concrete constellations of variable values pertaining to the building and the PV system). As such, the changes in the individual variable values cannot be directly observed form this Figure 48. Specifically, Figure 47 and Figure 48 also do not explicitly display the interesting and dynamic back and forth in the evolution of the evolving building-related variables and the PV system-related variables. To pursue this matter further, a smaller number of design states spread over the trajectory was randomly selected as depicted in Figure 49. This Figure entails again, for the smallest PV size class (18 kWp) and natural ventilation option, the trajectory of the design variants. The selected instanced highlighted in Figure 49 are further specified in Table 10, which includes variables related to both the building and the PV system. Thereby, building design variables pertain to the building (orientations E-W, SE-NW, S-N, SW-NE; glazing fraction of the façade in %) and the PV system (PV panel tilt; PV panel azimuth SE, S, SW).

It is instructive to consider the scope of reshuffling in the values of the different variables as represented in Table 1. Both the building's energy demand profile and the PV system's electricity generation profile change with each iteration, and these changes are reflected in the evolving value of the performance indicator (in this case, the payback time for the PV system). Consideration and analysis of this interdependency makes sense, if a building-integrated PV system is considered as a lasting component of a building project. Concurrent consideration of the designs of building and PV system does not mean that strict and undue constraints are imposed on the freedom in the selection of building design features. Interestingly, Figure 49 and Table 10 demonstrate that, for a certain fairly narrow value range of the performance indicator, multiple and diverse configurations of building design variables can provide similar levels of performance.

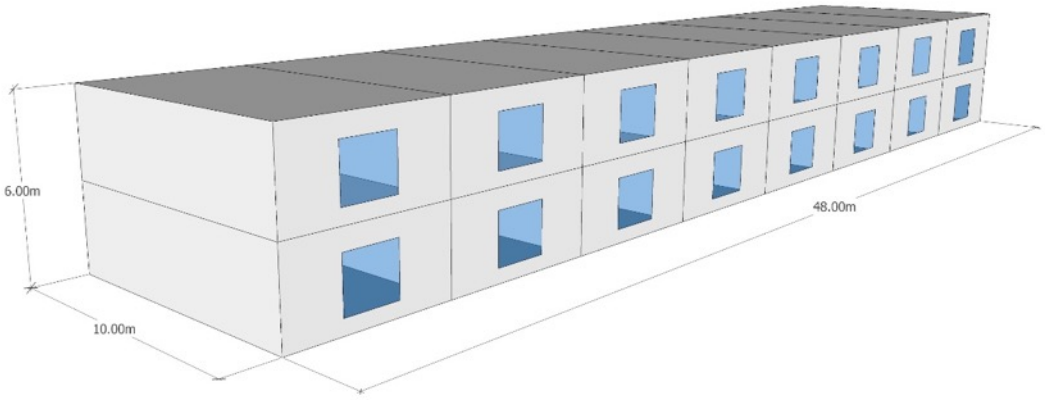

Figure 46 Schematic illustration of the eight-unit row house complex design located in the city of Vienna (Austria).

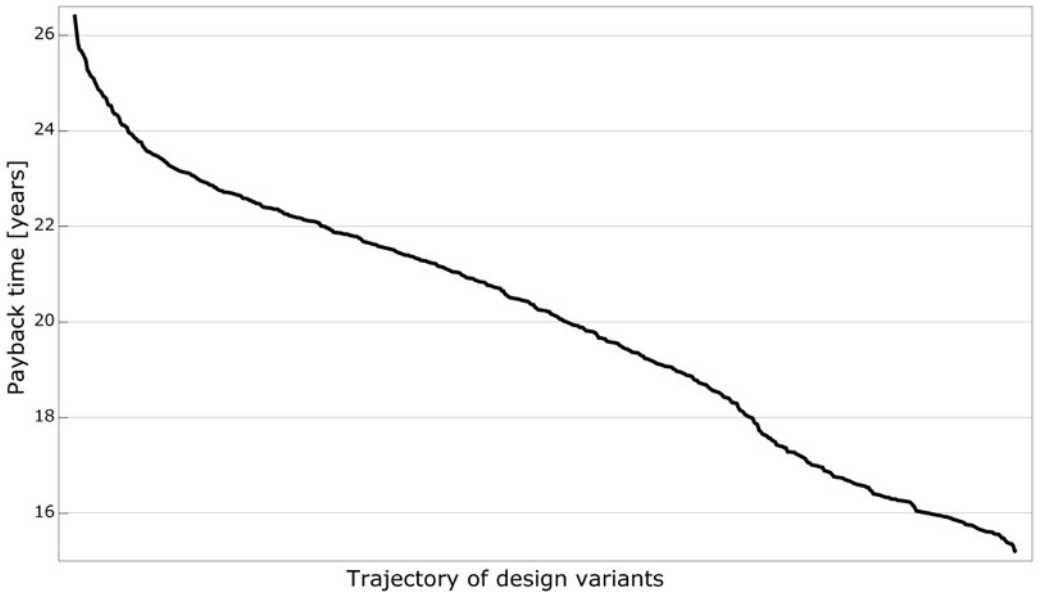

Figure 47 Estimated payback time for PV system's installation and maintenance cost plotted across the trajectory of the building and PV system variants.

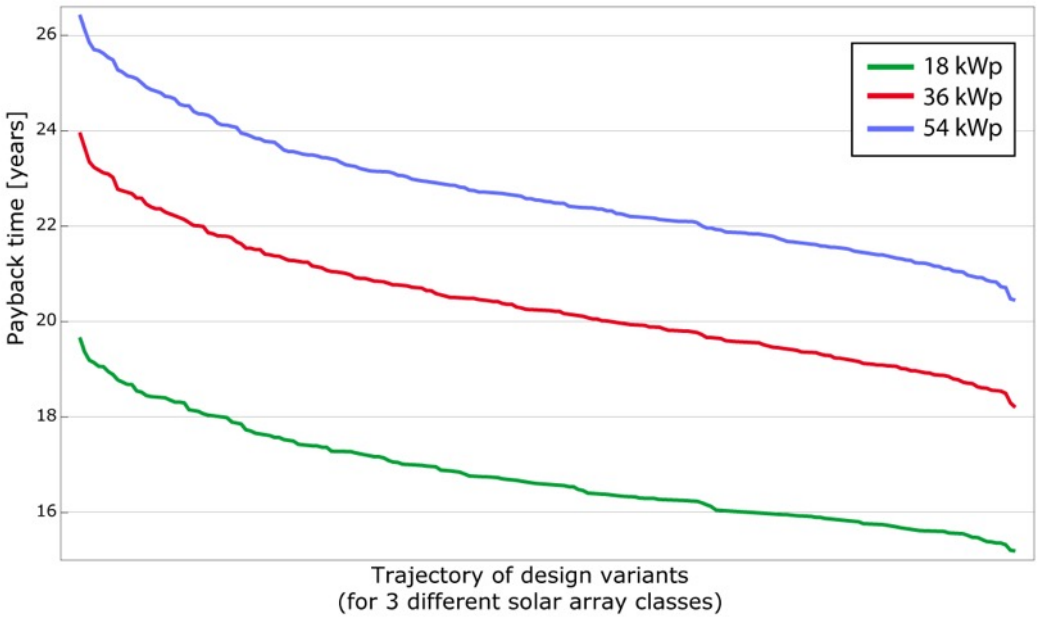

Figure 48 Estimated payback time for PV system's installation and maintenance cost plotted across the trajectory of the building and PV system variants shown for three distinct classes of PV installation sizes.

Table 10 State descriptions of the design variables regarding the building (orientations E-W, SE-NW, S-N, SW-NE; glazing fraction of the façade 20, 30, and 40 %) and the PV system (PV panel tilts 15, 30, 45 degrees; PV panel azimuth values SE, S, SW) for a selected number of design states along the trajectory shown in Figure 49. As with Figure 49, the states in this Table are arranged in descending order of the respective computed payback times in month.ec

		Building Orientation				Glazing			PV Tilt			PV Azimuth		
		E-W	SE-NW	S-N	SW-NE	20%	30%	40%	15°	30°	45°	SE	S	SW
Payback time [months]	236			✓		✓					✓	✓		
	228		✓				✓				✓	✓		
	224			✓		✓				✓		✓		
	221			✓		✓			✓			✓		
	217		✓			✓			✓			✓		
	215				✓		✓			✓		✓		
	212		✓					✓		✓		✓		
	211				✓		✓		✓			✓		
	208		✓			✓					✓			✓
	207		✓			✓			✓					✓
	206		✓			✓			✓				✓	
	204		✓			✓					✓		✓	
	203	✓				✓			✓					✓
	202				✓	✓				✓				✓
	201		✓				✓		✓					✓
	200				✓		✓		✓					✓
	199				✓	✓				✓			✓	
	198		✓					✓			✓			✓
	197	✓					✓				✓			✓
	196		✓				✓			✓			✓	
	196			✓				✓		✓			✓	
	195				✓			✓	✓					✓
	192		✓					✓		✓				✓
	192		✓					✓		✓			✓	
	191				✓			✓		✓			✓	
	187	✓						✓		✓			✓	

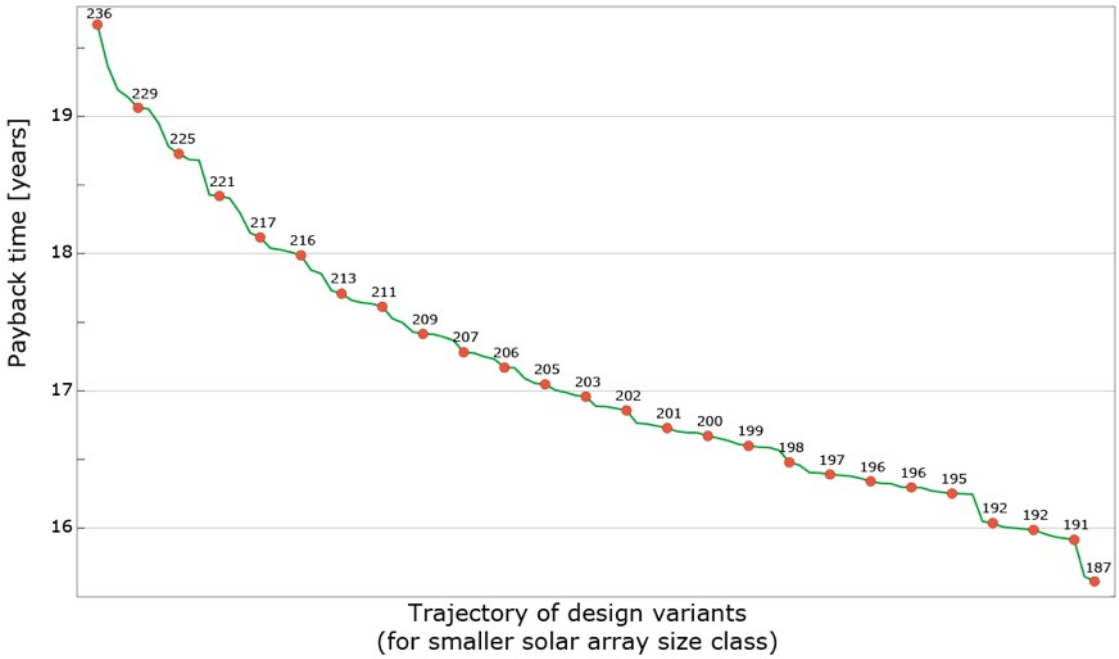

Figure 49 Estimated payback time for PV system's installation and maintenance cost plotted across the trajectory of the building and PV system variants shown for smallest PV installation size class and natural ventilation options. Shown are also the positions of a set of randomly selected states with the corresponding payback time expressed in months (see Table 10).

The proposed bi-directional computational approach to the concurrent performance analysis of building designs and respective building-integrated PV system installations method, allows not only for the exploration of the implications of buildings' energy demand profile for the configuration of the PV system, but also the other way around: The PV installation options can be explored in view of their potential implications for buildings' design features. As such, the proposed approach facilitated the parametric and iterative analysis of the both building design and PV configuration variables. Hence, those aspects of the building design could be identified, for which the temporal structure of energy demand and profile may be of relevance.

The presented approach involved wide spectrum of data both directly related to building and building performance but equally the technical specification of the equipment and various related financial aspects. The values of both the building design variables and the PV system variables (but potentially also the variables related to finance) could be adjusted within certain ranges. The results of operation of this computational framework verified its utility toward the concurrent exploration of the option space pertaining to building and PV system variables. Thereby, it could be demonstrated that a high level of overall performance

(expressed, for instance, in terms of the payback time for the PV system) can be achieved with very different constellations of both building design aspects and PV system options.

CHAPTER 4

Conclusion

4.1. Summary of contributions

The increasing presence and thus the importance of information and communications technology (ICT) in most scientific and technological domains and areas in the last few decades has led to an exponential growth in the volume of data. In order to make this overwhelming flow of data re-usable and to make it more accessible for both users and machines, the concept of ontologies and schemas was introduced.

The built environment domain equally benefits from the ICT advancements that enabled among others building information modelling, performance modelling, simulation and assessment. The building performance data (BPD) that drive the performance assessment applications is increasingly available, but due to multiplicity of sources, types and formats it is mostly syntactic, structural and semantic heterogeneous. To enable data interoperability that allow for development of methods, tools and applications that operate across multiple domains, scales and functions, it is necessary to establish versatile and robust ontologies.

Towards this end the building performance data (BPD) ontology was proposed. Such ontology is expected to facilitate data re-use and utilization in performance assessment applications.

This effort discussed the concept of ontologies and its current role across different fields of science and technology. It then focused on the built environment and gives an overview of the related ontologies that attempt to capture various aspects of the domain. Next, motivated by the recognized paucity in comprehensive representation of dynamic building performance related data, the BPD ontology and performance data schema was presented. The foundation for the developed ontology and schema was an extensive survey of common building performance indices, measures as well as monitoring data that explored the potential categorical groups, sources, and properties of the building performance-

related variables. The proposed schema was shown to capture key features of wide range of building performance variables in multiple performance domains, as illustrated by the examples presented.

The standard process and potential challenges concerning transformation of heterogeneous BPD were discussed. The said process where heterogeneous building performance related (measured or simulated) data is pre-processed, transformed, ontologized and stored, was exemplified on a large set of indoor monitoring data collected from an office building as well as monitored environmental data collected by a local weather station.

Such semantically enriched dataset was then put to a use in a series of basic to advanced building performance analysis and assessment scenarios. Specifically, potential of logical querying of ontologically enriched data for use with visualization and statistical analysis algorithms was tested. Moreover, a series of interfacing modules were developed that allowed for connection of performance data with a specific environmental design software. These interfaces were tested in various use case scenarios ranging from data visualization to advanced performance analysis concerning human comfort, daylight and solar radiation.

Note that the algorithms and interfaces make use of the formalized vocabulary and structure due to ontological processing. This formalization enables not only easy translation of data to conform to input format of existing analysis tools but also facilitates development of variety of performance applications that can make use of this defined data structure.

To further illustrate how the proposed data integration process can support multi-domain integrative and collaborative engineering efforts, an original web-based tool as well as two other application examples concerning building integrated PV systems were presented. These applications require and involve a range of data sources, such as location-specific climatic data (e.g. solar radiation), building related data (e.g. electricity demand profile), photovoltaic equipment specifications and related financial aspects (e.g. investment costs, maintenance costs, electricity pricing schemes). The applications make use of this data to assess input variations and deliver or visualize a range of technically and/or financially optimal options.

4.2. Future outlook

There is a recurring problem in the built environment field and there and it has two components. On the one hand, sufficient robust ontology instances that give a potential that was outlined in this effort must be developed and refined; on the other hand, these need to be adopted by the industry as well as wider community.

The ontology adaptation and development process can be time consuming and difficult (e.g. Industry Foundation Classes) as it often requires common interest (often financially motivated) and consent of multiple stakeholders involved. Similar challenges with regard to the implementation of ontologies and schemas for predominantly "static" aspect of the built environment concern ontologies with regard to "dynamic" aspects such as building

performance, maintenance or operation. In contrast to the purely market-driven use of ontologies (especially by companies with a high concentration of financial power), for example in search engines, social networks or knowledge systems, the ratio of demand to effort in built environment domain and more so in fields related to dynamic building data is not that high. Therefore, acquiring capital investment for the development of universal ontologies is a challenge.

The motivation to develop and more commonly implement a wider scope of ontologies related to the built environment can be nurtured by policy makers due to the growing awareness and need for a sustainable approach to the design, construction and operation of buildings. In some countries, this is already happening in relation to publicly procured buildings where IFC-driven BIM modeling is being enforced. Likewise, imposed requirements for ontologically systematized information processing should be useful in relation to the broader aspects of the domain of the built environment (and related domains) by enabling, among other things, scalable performance optimization, evaluation, and novel and comprehensive cross-domain solutions.

References

Project Related Publications

Notice: Major parts of this contribution have been formerly included in the following publications.

Mahdavi, A. and Wolosiuk, D. (2021). Ontologically streamlined data for building design and operation support (Chapter). Intelligent Environments - Advanced Systems for a Healthy Planet. Droege, P. (edt.), Elsevere. (To be published, 2021)

Wolosiuk, D., Schuss, M., and Mahdavi, A. (2021). Performance comparison of static and adjustable photovoltaic panels. Proceedings of the 17th IBPSA Conference, Bruges, Belgium, 1-3 September 2021.

Mahdavi, A., Wolosiuk, D., and Berger, C. (2021). A bi-directional approach to building-integrated PV systems configuration. 8th International Buildings Physics Conference 2021, Copenhagen, Denmark.

Wolosiuk, D. and A. Mahdavi, A. (2020a). Application of ontologically structured data for building performance analysis. Proceedings of the 11th annual Symposium on Simulation for Architecture & Urban Design (SimAUD). Vienna (Austria). 25-27 May 2020.

Wolosiuk, D., and Mahdavi, A. (2020b). Application of ontologically streamlined data for building performance analysis. Proceedings of the 13th European Conference on Product & Process Modelling (ECPPM 2021), 15-17 September 2021, Moscow, Russia

Mahdavi, A. and Wolosiuk D. (2019a). Integration of operational data in building information modelling: From ontology to application. CLIMA 2019 Congress. E3S Web Conf. Volume 111.

Mahdavi, A. and Wolosiuk D. (2019b). A Building Performance Indicator Ontology: Structure and Applications. 16th International IBPSA Conference.

Bibliography

Abecker A., van Elst L. (2009) Ontologies for Knowledge Management. In: Staab S., Studer R. (eds) Handbook on Ontologies. International Handbooks on Information Systems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-92673-3_32

Anderson, K. and Mikofski, M. (2020). Slope-Aware Backtracking for Single-Axis Trackers. Technical Report. NREL/TP-5K00-76626. National Renewable Energy Laboratory. Golden, CO (USA).

Ashburner, M., Ball, C. A., Blake et al., (2000). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nature genetics, 25(1), 25–29. https://doi.org/10.1038/75556

ASHRAE (2001). International Weather for Energy Calculations (IWEC Weather Files) User's Manual and CD-ROM. Atlanta, GA (USA).

Balaji, B., Bhattacharya, A., Fierro, G., Gao, J., Gluck, J., Hong, D., Johansen, A., Koh, J., Ploennigs, J., Agarwal, Y., Berges, M., Culler, D., R.K. Gupta, M.B. Kjærgaard, M. Srivastava, K. Whitehouse, (2018). Brick: metadata schema for portable smart building applications, Appl. Energy 226 (2018) 1273–1292.

Berners-Lee, T., Hendler, J.A., Ora, L. (2001). The SemanticWeb. Scientific American 284(5):34–43. [Online]. Available: https://www.scientificamerican.com/article/the-semantic-web [Accessed: January 2021].

Biermayr, P. et al. (2020). Innovative Energy Technologies in Austria: Market Development 2019. Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK). Vienna (Austria).

Bodenreider, O., Mitchell, J. A., & McCray, A. T. (2005). Biomedical ontologies. Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, 76–78. https://doi.org/10.1142/9789812704856_0016

Borrmann, A., Beetz, J., Koch, C., Liebich, T. & Muhic, Sergej. (2018). Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models. 10.1007/978-3-319-92862-3_5.

Brick Consortium (2021a). Brick Consortium Kickoff [Available at: https://brickschema.org/docs/Consortium_Kickoff_2021.pdf [Accessed: May 2021]

Brick Consortium (2021b). Brick Ontology Documentation. [Online]. Available: https://docs.brickschema.org [Accessed: February 2021]

buildingSMART (2021a). buildingSMART - The International Home of BIM [Online]. Available: http://buildingsmart.org [Accessed: May 2021].

buildingSMART (2021b). IFC4.3 Documentation. [Online]. Available: http://ifc43-docs.standards.buildingsmart.org [Accessed: May 2021].

BuildingSync, 2021. BuildingSync schema. [Online]. Available: https://buildingsync.net/ [Accessed: May 2021]

City of Vienna (2021). Green electricity systems or photovoltaic systems - funding application. [Online]. Available: https://www.wien.gv.at/amtshelfer/bauen-wohnen/energie/alternativenergie/oekostromanlagen.html [Accessed: April 2021]

Constantinou, N. (2017). A comprehensive multi-domain building performance indicator catalogue. Master thesis. Department of Building Physics and Building Ecology, TU Wien. Vienna (Austria).

Crawley, D.B., Hand, J.W., Lawrie. L.K. (1999). Improving the Weather Information Available to Simulation Programs. Proceedings of Building Simulation '99, Volume(II). Kyoto (Japan), September 1999.

Davies, J., Fensel, D., van Harmelen, F. (eds) (2002). Towards the Semantic Web: Ontology-driven Knowledge Management. ISBN: 978-0-470-84867-8. Wiley, London.

Davies, J., 2010. Lightweight Ontologies, in: Theory and Applications of Ontology: Computer Applications, R. Poli, M. Healy and A. Kameas, eds,Springer, Dordrecht, Netherlands, pp. 197–229.1272doi:10.1007/978-90-481-8847-5_9.1273.

DeGraw, J., Field-Macumber, K., Long, N., Goel, S. , BuildingSync® in Action : example implementations, vol. 2018, ACEEE Summer Study Energy Effic Build (2018), pp. 1-12

Domingue J., Fensel D., Hendler J.A., 2011. Introduction to the Semantic Web Technologies. In: Domingue J., FenselD., Hendler J.A. (eds) Handbook of Semantic Web Technologies. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-92913-0_1

Domingue, J., Fensel, D. and Hendler, J.A.(eds), 2011. Handbook of semantic web technologies, Springer, doi:10.1007/978-3-540-92913-0.

EnergyPlus, 2021. [Online]. Available: https://energyplus.net/ [Accessed: May 2021]

Esnaola-Gonzalez, I.; Díez, F.J. 2020. RESPOND Ontology Specification. [Online]. Available: https://respond-project.github.io/RESPOND-Ontology/respond/index-en.html [Accessed: January 2021].

ETSI, European Telecommunications Standards Institute (2020). SAREF: The Smart Applications REFerence ontology. [Online]. Available: https://saref.etsi.org/core/v3.1.1/ [Accessed: March 2021]

European Commission and TNO, (2015). Study on Semantic Assets for Smart Appliances Interoperability. [Online]. Available:

https://sites.google.com/site/smartappliancesproject/deliverables [Accessed: May 2021]

Eurostat, the statistical office of the European Union (2020a). Labour cost levels by NACE Rev. 2 activity. Electronic dataset. [Online]. Available: https://ec.europa.eu/eurostat/web/products-datasets/-/lc_lci_lev [Accessed: January 2021].

Eurostat, the statistical office of the European Union (2020b). Electricity prices for household consumers - bi-annual data (from 2007 onwards). Electronic dataset. [Online]. Available: https://ec.europa.eu/eurostat/web/products-datasets/-/nrg_pc_204 [Accessed: January 2021].

Fanger, P.O., (1970). Thermal Comfort Analysis and Applications in Environmental Engineering. Mcgraw-Hill, New York.

Foliente, Greg. (2000). Developments in performance-based building codes and standards. Forest Products Journal. 50. 12-21.

Fu, R., Feldman, D. and Margolis, R. (2018). U.S. Solar Photovoltaic System Cost Benchmark: Q1 2018. NREL/TP-6A20-72399. National Renewable Energy Laboratory. Golden, CO (USA).

Genesereth, M. R., & Nilsson, N. J. (1987). Logical Foundations of Artificial Intelligence. San Mateo, CA: Morgan Kaufmann Publishers.

Gómez-Pérez, A., Fernández-López, M., Corcho, O., (2007). Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web. (Advanced Information and Knowledge Processing). Springer-Verlag, Berlin, Heidelberg.

Gruber T. (2009) Ontology. In: Liu. L., Özsu, M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_1318

Guarino N., Oberle D., Staab S. (2009) What Is an Ontology?. In: Staab S., Studer R. (eds) Handbook on Ontologies. International Handbooks on Information Systems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-92673-3_0

Guarino, N. and Giaretta, P., (1995). Ontologies and Knowledge Bases. In: Towards Very Large Knowledge Bases, IOS Press, Amsterdam, 1-2.

Hartel, F. W., de Coronado, S., Dionne, R., Fragoso, G., & Golbeck, J. (2005). Modeling a description logic vocabulary for cancer research. Journal of biomedical informatics, 38(2), 114–129. https://doi.org/10.1016/j.jbi.2004.09.001

Hay, J.E. and Davies, J.A. (1980). Calculations of the solar radiation incident on an inclined surface. In Hay, J.E., Won, T.K. Proceedings of First Canadian Solar Radiation Data Workshop. Ministry of Supply and Services. Toronto (Canada).

Hipp, R.D., (2021). SQLite. [Online]. Available: https://www.sqlite.org/index.html. [Accessed: April 2021].

Hitzler, P., M. Krötzsch, B. Parsia, P.F. Patel- Schneider and S. Rudolph, (2009) OWL 2 web ontology language primer. [Online]. Available: https://www.w3.org/TR/owl2-primer/ [Accessed: February 2021]

Holmgren, W.F., Hansen, C.W., and, Mikofski, M.A. (2018). pvlib-python: a python package for modelling solar energy systems. Journal of Open Source Software, 3(29), 884.

ISO 16739 (2013). Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries. International Organization for Standardization, Geneva, Switzerland

International Organization for Standardization, (2018). ISO 16739-1:2018 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries — Part 1: Data schema

Jäger-Waldau, A. (2019). PV Status Report 2019 (EUR 29938 EN). Publications Office of the European Union, Luxembourg, 2019, ISBN 978-92-76-12608-9.

JetBrains, (2021). The Python IDE. [Online]. Available: https://www.jetbrains.com/pycharm/ [Accessed January 2021]

King, D. et al. (2004). Sandia Photovoltaic Array Performance Model, SAND Report 3535, Sandia National Laboratories, Albuquerque, NM.

Lassila O, Swick RR. Resource description framework (RDF) model and syntax specification.

Lassila, O., Swick, R. (Eds.), (1999). Resource description framework (RDF): model and syntax specification. Technical report, W3C, 1999. W3C Recommendation 1999-02-22, [Online]. Available: http://www.w3.org/TR/REC-rdf-syntax/ [Accessed May 2021].

Mahdavi A., (2011). People in Building Performance Simulation. In Hensen, J.L.M., Lamberts R. Building Performance Simulation for Design and Operation. Taylor & Francis Group. New York (USA).

Mahdavi, A., 1998. Steps to general theory of habitability, Human Ecology Review 5(1).

Mahdavi, A., Bachinger, J., and Suter, G. (2005). Toward a unified information space for the specification of building performance simulation results. In: Building Simulation 2005. 9th International IBPSA Conference, Aug. 15-18, Montreal (Canada). Editors: I. Beausoleil-Morrison, M. Bernier. p.671-676.

Mahdavi, A., Glawischnig, S., Schuss, M., Tahmasebi, F., and Heiderer, A. (2016). Structured Building Monitoring: Ontologies and Platform. Proceedings of ECPPM 2016: The 11th European Conference on Product and Process Modelling. Limassol (Cyprus)

Mahdavi, A., Taheri, M. (2017). An Ontology for Building Monitoring. Journal of Building Performance Simulation. 10(5-6), 499-508. DOI:10.1080/19401493.2016.1243730.

Mahdavi, A., Taheri, M., Schuss, M., Tahmasebi, F., and Glawischnig, S. (2018). Structured Building Data Management: Ontologies, Queries, and Platforms. Exploring Occupant Behavior in Buildings. Publisher: Springer. Editors: Andreas Wagner, William O'Brien, Bing Dong. DOI: 10.1007/978-3-319-61464-9_10.

Marion, W. and Dobos, A. (2013). Rotation Angle for the Optimum Tracking of One-Axis Trackers. NREL Technical Report. NREL/TP-6A20-58891. National Renewable Energy Laboratory. Golden, CO (USA).

Pritoni, M., Paine, D., Fierro, G., Mosiman, C., Poplawski, M., Saha, A., Bender, J., Granderson, J., (2021). Metadata Schemas and Ontologies for Building Energy Applications: A Critical Review and Use Case Analysis. Energies14, no. 7: 2024. https://doi.org/10.3390/en14072024

Project Haystack, (2021a). Project Haystack - An Open Source initiative to streamline working with IoT Data. [Online]. Available: https://project-haystack.org. [Accessed May 2021].

Project Haystack, (2021b). Implementing Project Haystack - Applying Haystack Tagging for a sample building. [Online]. Available: https://project-haystack.org/file/28/Reference-Implementation--Applying-Haystack-Tagging-for-a-Sample-Building.pdf [Accessed May 2021].

Rasmussen, M.H., Pauwels, P., Hviid, C.A., Karlshøj, J., (2017a). Proposing a Central AEC Ontology That Allows for Domain Specific Extensions, in: Proceedings of the Joint Conference on Computing in Construction, Vol. 1, Heriot-Watt University, Heraklion, Crete, Greece. ISBN 978-0-9565951-6-4.doi:10.24928/jc3-2017/0153.

Rasmussen, M.H., Pauwels, P., Lefrançois, M., Schneider, G.F., Hviid, C., Karlshøj, J., (2017b). Recent changes in the Building Topology Ontology, in: 5th Linked Data in Architecture and Construction Workshop, Dijon, France, 2017b. doi:10.13140/RG.2.2.32365.28647.

Rasmussen, Mads Holten & Lefrançois, Maxime & Schneider, Georg & Pauwels, Pieter. (2020). BOT: the Building Topology Ontology of the W3C Linked Building Data Group. Semantic Web. 10.3233/SW-200385.

Reda, I. and Andreas, A., (2004). Solar position algorithm for solar radiation applications. Solar Energy, vol. 76, no. 5, pp. 577-589, 2004.

RIS, The Legal Information System of the Republic of Austria (2020). Ökostrom-Einspeisetarifverordnung 2018 – ÖSET-VO 2018. BGBl. II Nr. 408/2017.

Rosse, C., and Mejino, J.L.V., (2007). The Foundational Model of Anatomy Ontology. In

A. Burger, D. Davidson, and R. Baldock, editors, Anatomy Ontologies for Bioinformatics: Principles and Practice, volume 6, pages 59-117, London, Springer.

Rubin, D.L., Nigam H. Shah, Natalya F. Noy, (2008). Biomedical ontologies: a functional perspective, Briefings in Bioinformatics, Volume 9, Issue 1, January 2008, Pages 75–90,

Sánchez, D.M., Cavero, J.M., Martínez, E.M., (2007). The Road Toward Ontologies. In: Sharman R., Kishore R., Ramesh R. (eds) Ontologies. Integrated Series in Information Systems, vol 14. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-37022-4_1

Schenck, D. A., and Wilson, P. R. (1993). Information Modeling the EXPRESS Way, Oxford University Press.

Staab, S., Studer, R., (2009). Handbook on Ontologies. ISBN 978-3540709992 Springer, Berlin.

Staab, S., Studer, R., Schnurr, H.P., Sure-Vetter, Y., (2001). Knowledge Processes and Ontologies. IEEE Intelligent Systems. 16. 26-34. 10.1109/5254.912382.

Stein, J.S., (2012) The photovoltaic Performance Modeling Collaborative (PVPMC). Proceedings 38th IEEE Photovoltaic Specialists Conference. Austin, TX (USA). 3-8 June 2012.

Taniar,D. and Wenny Rahayu, J., (2006). Web Semantics Ontology. IGI Global, USA.

Tregenza, P. R. 1987. Subdivision of the Sky Hemisphere for Luminance Measurements, Lighting Research and Technology 19:13-14

Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, methods and applications. The Knowledge Engineering Review, 11(2), 93-136. doi:10.1017/S0269888900007797

Uschold, M. and King, M., (1995). Towards a Methodology for building ontologies. In: Skuce D, ed. IJCAI'95m Workshop on Basic Ontological Issue in Knowledge Sharing. Montreal, 6.1-6.10.

W3C, (2021). Linked Building Data Community Group. [Online]. Available: https://www.w3.org/community/lbd/ [Accessed May 2021]

Wilcox, S. and W. Marion. (2008). User's Manual for TMY3 Data Sets. NREL/TP-581-43156. National Renewable Energy Laboratory. Golden, CO (USA).

Wilde, P., 2018. Building Performance Analysis. 10.1002/9781119341901.

Yan, D., Hong, T., Dong, B., Mahdavi, A., D'Oca, S., Gaetani, I., Feng, X. (2017). IEA EBC Annex 66: Definition and Simulation of Occupant Behavior in Buildings. Energy and Buildings. 156. 10.1016/j.enbuild.2017.09.084.

List of Tables

List of Figures

Appendix

The following appendix contains a selection of source code written in the Python programming language, which was created in the course of this research project development.

```
# ============================
# BPI Sensor db files to BPD schema
# ============================


import numpy as np
import h5py
import sqlite3
from pandas.io import sql
from datetime import datetime

# Extract necessary data from CSV file (attributes etc)
my_data =
np.genfromtxt('/Users/dw/PycharmProjects/37project/Ontology/BPIdb_to_HDF5_sensorlist_ALL_2019_v2.csv',
                        delimiter=',', dtype=None, encoding='UTF-8')

# Set hdf5 file name
HDF5fileName = '/Users/dw/PycharmProjects/37project/Ontology/BPI_office_fromCSV_2019_new_190903.hdf5'

for x in my_data[2:]:  # Iterate through every row (starting from third) in the my_data array (extracted
from CSV)

    if len(x[0]) > 10:  # Check if the file name exists if entry exists(not an empty row)
        print('Currently processing: ' + x[0])
        sqlDBname = x[0]
        sqlDBsensorType = x[0].split('_')[0]

        sensorName = x[0][:-11]  # Cut off last 11 characters of the DB name string

        # Create your connection.
        cnx = sqlite3.connect(
            '/Users/dw/PycharmProjects/37project/Ontology/mySQLdata/' + sqlDBsensorType + '/' + sqlDBname)

        # read the result of the SQL query into a DataFrame
        data = sql.read_sql("SELECT `_rowid_`,* FROM `data` ORDER BY `Date` ASC;", cnx)
        cnx.close()

        # CONVERT DATE STRINGS TO NUMERICAL VALUES and put to numpy array:
        date = np.array([int(datetime.fromisoformat(line).timestamp()) for line in
data['Date'].values]).astype(
            'uint32')

        # IMPORT SENSOR VALUES TO NUMPY ARRAY
        magnitude = np.array(data[sensorName].values.astype(x[27]))  # x[27] - Data type in the csv file

        # CREATE NEW VARIABLE, ADD ATTRIBUTES, STORE MAGNITUDE & TIME DATA
        f = h5py.File(HDF5fileName, 'a')  # Open read-write (create if doesn't exist)

        # Creates group (if not existent already) Category/SubCategory/Name of indicator
        variable = f.require_group('BPIcategories/' + x[1] + '/' + x[2] + '/' + x[4] + '/' + x[3])
        if x[4] != '': variable.attrs['Name'] = str(x[4])
        if x[5] != '': variable.attrs['Type'] = str(x[5])
        variable['Magnitude'] = magnitude
        if x[7] != '': variable.attrs['Direction'] = np.array(x[7].split()).astype(np.float32)
        if x[8] != '': variable.attrs['Unit'] = str(x[8])

        # Spatial Domain Attributes
        variableSD = f.create_group('BPIcategories/' + x[1] + '/' + x[2] + '/' + x[4] + '/' + x[3] +
'/SpatialDomain')
        if x[9] != '': variableSD.attrs['Point'] = np.array(x[9].split()).astype(np.float32)
        if x[10] != '': variableSD.attrs['Plane'] = str(x[10])
        if x[11] != '': variableSD.attrs['Volume'] = str(x[11])
        if x[12] != '': variableSD.attrs['TopologicalReference'] = str(x[12])
        if x[13] != '': variableSD.attrs['AggregationMethod'] = str(x[13])
        if x[14] != '': variableSD.attrs['GridSize'] = np.array(x[14].split()).astype(np.uint32)

        # Temporal Domain Attributes
        variableTD = f.require_group('BPIcategories/' + x[1] + '/' + x[2] + '/' + x[4] + '/' + x[3] +
'/TemporalDomain')
        variableTD['TimeStamp'] = date
        if x[16] != '': variableTD.attrs['Duration'] = float(x[16])
        if x[17] != '': variableTD.attrs['TimeStep'] = str(x[17])
        if x[18] != '': variableTD.attrs['AggregationMethod'] = str(x[18])

        # Frequency Domain Attributes
        variableFD = f.require_group(
            'BPIcategories/' + x[1] + '/' + x[2] + '/' + x[4] + '/' + x[3] + '/FrequencyDomain')
        if x[19] != '': variableFD.attrs['Range'] = np.array(x[19].split()).astype(np.uint32)
        if x[20] != '': variableFD.attrs['Band'] = np.array(x[20].split()).astype(np.uint32)
        if x[21] != '': variableFD.attrs['Weighting'] = str(x[21])
        if x[22] != '': variableFD.attrs['AggregationMethod'] = str(x[22])

        # AGENT
        variableAG = f.require_group('BPIcategories/' + x[1] + '/' + x[2] + '/' + x[4] + '/' + x[3] +
'/Agent')
        if x[23] != '': variableAG.attrs['AgentID'] = str(x[23])
```

```
        # NOTES
        variableNT = f.require_group('BPIcategories/' + x[1] + '/' + x[2] + '/' + x[4] + '/' + x[3] +
'/Notes')
        if x[24] != '': variableNT.attrs['DataSourceCategory'] = str(x[24])
        if x[25] != '': variableNT.attrs['DataSourceID'] = str(x[25])
        if x[26] != '': variableNT.attrs['DerivationMethodDetails'] = str(x[26])
        if x[28] != '': variableNT.attrs['DerivationMethodDetails'] = str(x[28])

        f.close()
```

```
################################
# SKYSCANNER DATA Files to BPD #
################################

import pandas
import os
import re
import numpy as np
from datetime import datetime, timedelta

SkyscannerDataDir = '/Users/dw/PycharmProjects/37project/Ontology/SKY_SCANNER/'

fileList = []
dateList = []

for aFile in os.listdir(SkyscannerDataDir):
    if aFile.endswith(".csv"):
        if not aFile.startswith('.'):
            fileList = np.append(fileList, aFile)
            match = re.search(r'\d{2}\d{2}\d{2}', aFile)

fileList.sort()

# Get all data and timestamps
nrOfDataYears = 9
magnitudeSR = np.empty((18300 * nrOfDataYears, 145), dtype=float)
magnitudeSL = np.empty((18300 * nrOfDataYears, 145), dtype=float)
magnitudeSPdirection = np.empty(18300 * nrOfDataYears, dtype=float)
magnitudeSPelevation = np.empty(18300 * nrOfDataYears, dtype=float)
date = np.empty(18300 * nrOfDataYears, dtype=int)
counter = 0
for aFile in fileList:

    data = pandas.read_csv(SkyscannerDataDir + aFile, header=8)
    data.rename(
        columns={'Unnamed: 0': 'Reading', 'Unnamed: 1': 'MeasurementStartTime', 'Unnamed: 2':
'MeasurementStopTime',
                 'Unnamed: 148': 'SunDirection', 'Unnamed: 149': 'SunElevation'}, inplace=True)

    # transpose matrix:
    data = data.T
    readings = data.reindex(np.concatenate(
        (["Reading", "MeasurementStartTime", "MeasurementStopTime"],
         np.arange(16, 31, 1), np.arange(1, 16, 1),
         np.arange(45, 30, -1), np.arange(60, 45, -1),
         np.arange(73, 85, 1), np.arange(61, 73, 1),
         np.arange(96, 84, -1), np.arange(108, 96, -1),
         np.arange(118, 127, 1), np.arange(109, 118, 1),
         np.arange(132, 126, -1), np.arange(138, 132, -1),
         np.arange(142, 145, 1), np.arange(139, 142, 1),
         np.arange(145, 146, 1), ["SunDirection", "SunElevation"])))

    matchDay = re.search(r'\d{2}\d{2}\d{2}', aFile)

    for x in range(0, len(readings.columns)):
        if readings[x][0] == "L":
            # print(readings[x])
            matchTime = re.search(r'\d{2}:\d{2}:\d{2}', readings[x][1])

            date[counter] = int((datetime.strptime(matchDay.group() + matchTime.group(),
'%y%m%d%H:%M:%S') - timedelta(
                hours=1)).timestamp())  # the time stored in the skyscanner files is in utc +1

            print(datetime.strptime(matchDay.group() + matchTime.group(), '%y%m%d%H:%M:%S') -
timedelta(hours=1))
            magnitudeSL[counter] = np.array([readings[3:148][x]])
            magnitudeSPdirection[counter] = np.array([readings[148:149][x]])
            magnitudeSPelevation[counter] = np.array([readings[149:150][x]])

            # match.group()+match2.group() #join two regex match queries
        else:
            # print(a[x])
            magnitudeSR[counter] = np.array([readings[3:148][x]])
            counter = counter + 1

# In case the number of measurements was smaller than declared 18300 * nr of years - cut "empty" rows
out.
date = date[0:counter]
magnitudeSL = magnitudeSL[0:counter]
magnitudeSR = magnitudeSR[0:counter]
magnitudeSPdirection = magnitudeSPdirection[0:counter]
magnitudeSPelevation = magnitudeSPelevation[0:counter]

np.save('magnitudeSL.npy', magnitudeSL)
np.save('magnitudeSR.npy', magnitudeSR)
np.save('dates.npy', date)
```

```
np.save('SunDirection.npy', magnitudeSPdirection)
np.save('SunElevation.npy', magnitudeSPelevation)

# 2nd version
# CREATE NEW VARIABLE, ADD ATTRIBUTES, STORE MAGNITUDE & TIME DATA
import h5py

HDF5fileName = "/Users/dw/PycharmProjects/37project/Ontology/BPI_office_fromCSV_2019_new_190903.hdf5"
f = h5py.File(HDF5fileName, 'a')  # Open read-write (create if doesn't exist)

# Creates group (if not existent already) Category/SubCategory/Name of indicator
variableSR = f.require_group(
    'BPIcategories/ExternalConditions/SkyRadiance/SkyScanner')  # Category/subCategory,variable
Category   Sub_Category   Variable

variableSR.attrs['Name'] = "SkyRadiance"
variableSR.attrs['Type'] = "quantitative"
variableSR['Magnitude'] = magnitudeSR
variableSR.attrs['Direction'] =
          np.array([np.concatenate((    np.repeat(6, 30), np.repeat(18, 30),
                                        np.repeat(30, 24), np.repeat(42, 24), np.repeat(54, 18),
                                        np.repeat(66, 12), np.repeat(78, 6), [90])),

          np.concatenate((np.arange(0, 181, 12), np.arange(-168, 0, 12),
                          np.arange(0, 181, 12), np.arange(-168, 0, 12),
                          np.arange(0, 181, 15), np.arange(-165, 0, 15),
                          np.arange(0, 181, 15), np.arange(-165, 0, 15),
                          np.arange(0, 181, 20), np.arange(-160, 0, 20),
                          np.arange(0, 181, 30), np.arange(-150, 0, 30),
                          np.arange(0, 181, 60), np.arange(-120, 0, 60),
                          [0]
                          ))
          ], np.int16)

variableSR.attrs['Unit'] = "W/(m^2*sr)"
variableSRSD =
f.require_group('BPIcategories/ExternalConditions/SkyRadiance/SkyScanner/SpatialDomain')
# variableSRSD.attrs['Point'] =
# variableSRSD.attrs['Plane'] =
variableSRSD.attrs['Volume'] = "TU Wien"
variableSRSD.attrs['TopologicalReference'] = "Tower"
# variableSRSD.attrs['AggregationMethod'] =
# variableSRSD.attrs['GridSize'] =

# Temporal Domain Attributes
variableSRTD =
f.require_group('BPIcategories/ExternalConditions/SkyRadiance/SkyScanner/TemporalDomain')
variableSRTD['TimeStamp'] = date
# variableSRTD.attrs['Duration'] =
variableSRTD.attrs['TimeStep'] = "15min"
# variableSRTD.attrs['AggregationMethod'] =

# NOTES
variableSRNT = f.require_group('BPIcategories/ExternalConditions/SkyRadiance/SkyScanner/Notes')
variableSRNT.attrs['DataSourceCategory'] = "Sensor"
variableSRNT.attrs['DataSourceID'] = "SkyScanner"
# variableNT.attrs['DerivationMethodDetails'] =
variableSRNT.attrs[
    'Remarks'] = "Variable's Direction attribute describes Altitude and Azimuth of the measurement
sample taken by sensor conforming with" \
                 " Tregenza sky subdivision."

# LUMINANCE
# CREATE NEW VARIABLE, ADD ATTRIBUTES, STORE MAGNITUDE & TIME DATA

# Creates group (if not existent already) Category/SubCategory/Name of indicator
variableSL = f.require_group('BPIcategories/ExternalConditions/SkyLuminance/SkyScanner')

variableSL.attrs['Name'] = "SkyLuminance"
variableSL.attrs['Type'] = "quantitative"
variableSL['Magnitude'] = magnitudeSL
variableSL.attrs['Direction'] = variableSR.attrs['Direction']
variableSL.attrs['Unit'] = "kcd/m^2"

variableSLSD =
f.require_group('BPIcategories/ExternalConditions/SkyLuminance/SkyScanner/SpatialDomain')
# variablSLSD.attrs['Point'] =
# variableSLSD.attrs['Plane'] =
variableSLSD.attrs['Volume'] = "TU Wien"
variableSLSD.attrs['TopologicalReference'] = "Tower"
# variableSRSD.attrs['AggregationMethod'] =
# variableSRSD.attrs['GridSize'] =

# Temporal Domain Attributes
variableSLTD =
f.require_group('BPIcategories/ExternalConditions/SkyLuminance/SkyScanner/TemporalDomain')
```

```
variableSLTD['TimeStamp'] = date
# variableSLTD.attrs['Duration'] =
variableSLTD.attrs['TimeStep'] = "15min"
# variableSLTD.attrs['AggregationMethod'] =

# NOTES
variableSLNT = f.require_group('BPIcategories/ExternalConditions/SkyLuminance/SkyScanner/Notes')
variableSLNT.attrs['DataSourceCategory'] = "Sensor"
variableSLNT.attrs['DataSourceID'] = "SkyScanner"
# variableSLNT.attrs['DerivationMethodDetails'] =
variableSLNT.attrs[
    'Remarks'] = "Variable's Direction attribute describes Altitude and Azimuth of the measurement
sample taken by sensor conforming with" \
                 " Tregenza sky subdivision."

f.close()
```

```
# ================================================
# Find object by attribute and plot v2
# ================================================

import h5py
import matplotlib.pyplot as plt
from h5py import File
import time
from datetime import datetime

f: File = h5py.File('/Users/dw/PycharmProjects/37project/Ontology/HDFs/BPI_office.hdf5', 'r')

# Attributes:
Xpos = 4
Ypos = 4.5
startDate = int(datetime.fromisoformat('2017-01-01 00:00:00').timestamp())
stopDate = int(datetime.fromisoformat('2017-12-31 23:59:59').timestamp())

a = []


def findspecial2(name, obj):
    if obj.attrs.get('point') is not None and obj.attrs.get('point')[0] == Xpos and
obj.attrs.get('point')[1] == Ypos:
        # print(obj)
        a.append(obj.parent)
        print(obj.parent)


t0 = time.time()
f.visititems(findspecial2)
t1 = time.time()


def plotGraphs(VariableList):
    for item in VariableList:
        print(item.name)
        dsetX = item['TemporalDomain/timeStamp']
        dsetY = item['magnitude']

        index = (dsetX.value > startDate) & (dsetX.value < stopDate)
        # extract indexed data and stor in a variable, close HDF5 file
        dsetY = dsetY.value[index]
        dsetX = dsetX.value[index]

        print('Dataset min date: ' + datetime.fromtimestamp(int(dsetX[0])).strftime('%Y-%m-%d
%H:%M:%S'))
        print('Dataset max date: ' + datetime.fromtimestamp(int(dsetX[-1])).strftime('%Y-%m-%d
%H:%M:%S'))
        plt.plot(dsetX[:], dsetY[:])
        # plt.title(item.name)
        plt.title(item.attrs.get('name') + 'trend')
        # plt.show()
        # plt.savefig(item.attrs.get('name')+'.pdf',bbox_inches='tight')
        plt.savefig('lin_' + item.attrs.get('name') + '.pdf', bbox_inches='tight')
        plt.show()
        plt.hist(dsetY, 50)
        plt.title(item.attrs.get('name') + ' frequency distribution')
        plt.savefig('hist_' + item.attrs.get('name') + '.pdf', bbox_inches='tight')
        plt.show()

        fig1, ax1 = plt.subplots()
        ax1.set_title(item.attrs.get('name') + 'box plot')
        ax1.boxplot(dsetY)
        plt.show()
        fig1.savefig('box_' + item.attrs.get('name') + '.pdf', bbox_inches='tight')


plotGraphs(a)
f.close()
# ================================================
```

```
#######################################
# PV explorer tool; web- Application demo
# key components:
# https://github.com/pvlib/pvlib-python
# https://dash.plotly.com
#######################################

import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.graph_objs as go
import pandas as pd
from dash.dependencies import State, Input, Output
from datetime import datetime as dt
import plotly.express as px
import pvlib
import glob, os


os.chdir("/Users/dw/PycharmProjects/37project/Ontology/PV_panels/Energy_Use_Profiles")
print("Available Irradiance data sources:")

n_clicks_glob = 0
# scan for energy use profiles
eleConsumptionProfiles = []
for file in glob.glob("*.csv"):
    # global irradiance_data_sources
    if file != None:
        eleConsumptionProfiles.append(file)
    print(file.replace(".csv", ""))

# scan for epw weather files
os.chdir("/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files")
weather_data_sources = []
for file in glob.glob("*.epw"):
    if file != None:
        weather_data_sources.append(file)
    print(file.replace(".epw", ""))

irradiance_model = 'haydavies'
irradiance_data_source = '/USA_TX_Austin.722540_TMY2.epw'

params = pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm'][
    'open_rack_glass_glass']  # https://pvlib-
python.readthedocs.io/en/stable/generated/pvlib.temperature.sapm_cell.html
PV_modules_list = pvlib.pvsystem.retrieve_sam('SandiaMod')
sapm_inverters_list = pvlib.pvsystem.retrieve_sam('CECInverter')


def instalation_costs_PV_area(panelArea, divider):
    instalationCosts = (269.09 * panelArea + 1319.4) / divider

    return instalationCosts


app = dash.Dash(
    __name__,
    meta_tags=[{"name": "viewport", "content": "width=device-width, initial-scale=1"}],
)

server = app.server
app.config.suppress_callback_exceptions = True

surface_azimuths = [90, 105, 120, 135, 150, 165, 180, 195, 210, 225, 240, 255, 270]
surface_tilts = [0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90]
panel_counts = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
inverter_price_index = {'high_end': 0.17, 'mainstream': 0.12, 'low_cost': 0.05}


def generate_cost_plot(inverter_price_index_type, dd_select_module, dd_select_inverter,
energy_export_price,
                       energy_import_price, cost_function,
                       start, end, weather_file, electricity_use_profile_file,
electricity_profile_element,
                       investment_cycle):
    PV_lifespan = investment_cycle  # TODO:fix naming
    global results
    results = pd.DataFrame()
    naive_times = pd.date_range(start=start, end=end, freq='1h')
    global system
    system = {'module': PV_modules_list[dd_select_module], 'inverter':
sapm_inverters_list[dd_select_inverter],
              'surface_azimuth': surface_azimuths, 'surface_tilt': surface_tilts, 'panel_count':
panel_counts}

    # prepare epw
```

```
    epw_file_path = '/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files/' +
weather_file
    epw_data = pvlib.iotools.read_epw(epw_file_path, coerce_year=2019)

    # extract needed data from epw
    temp_air = epw_data[0]['temp_air']
    wind_speed = epw_data[0]['wind_speed']

    coordinates = [(epw_data[1]['latitude'], epw_data[1]['longitude'], epw_data[1]['city'],
epw_data[1]['altitude'],
                    epw_data[1]['TZ'])]

    AnnnualHourlyEnergyUseProfileFile =
'/Users/dw/PycharmProjects/37project/Ontology/PV_panels/Energy_Use_Profiles/' +
electricity_use_profile_file
    eleConsumptionProfile = pd.read_csv(AnnnualHourlyEnergyUseProfileFile,
                                        index_col='Date/Time', parse_dates=True)
    eleConsumptionProfile[electricity_profile_element].groupby(
        eleConsumptionProfile[electricity_profile_element].index.hour).sum()
    loadProfile = eleConsumptionProfile[electricity_profile_element]
    loadProfile = loadProfile.tz_localize(int(epw_data[1]['TZ']) * 60 * 60)

    for latitude, longitude, name, altitude, timezone in coordinates:
        times = naive_times.tz_localize(int(timezone) * 60 * 60)  # localizes to GMT

        # times = naive_times.tz_localize(None)
        solpos = pvlib.solarposition.get_solarposition(times, latitude, longitude)
        dni_extra = pvlib.irradiance.get_extra_radiation(times)
        airmass = pvlib.atmosphere.get_relative_airmass(solpos['apparent_zenith'])
        pressure = pvlib.atmosphere.alt2pres(altitude)
        am_abs = pvlib.atmosphere.get_absolute_airmass(airmass, pressure)
        tl = pvlib.clearsky.lookup_linke_turbidity(times, latitude, longitude)
        cs = pvlib.clearsky.ineichen(solpos['apparent_zenith'], am_abs, tl, dni_extra=dni_extra,
altitude=altitude)

        temp_final_panel_count = []
        temp_final_azimuth = []
        temp_final_tilt = []
        temp_final_annual_total_irradiance = []
        temp_final_annual_total_AC = []
        temp_final_annual_balance_kWh_deficit = []
        temp_final_annual_balance_kWh_surplus = []
        temp_final_annual_ac_deficit_cost_EUR = []
        temp_final_annual_ac_balance_EUR = []

        for azim in system['surface_azimuth']:

            temp_irr_val = []  # to get all annual values from a single azimuth
            temp_AC_val = []  # to get all annual AC output per selected panel values from a single
azimuth
            temp_deficiency = []
            temp_surplus = []
            temp_deficiency_series = []
            for tilt in surface_tilts:
                aoi = pvlib.irradiance.aoi(tilt, azim, solpos['apparent_zenith'], solpos['azimuth'])
                total_irradiance = pvlib.irradiance.get_total_irradiance(tilt,
                                                                         azim,
                                                                         solpos['apparent_zenith'],
                                                                         solpos['azimuth'],
                                                                         epw_data[0]['dni'],
epw_data[0]['ghi'],
                                                                         epw_data[0]['dhi'],
                                                                         dni_extra=dni_extra,
                                                                         model='haydavies')
                temps =
pvlib.pvsystem.temperature.sapm_cell(total_irradiance['poa_global'].tz_convert(timezone),
                                                         temp_air, wind_speed, **params)
                effective_irradiance = pvlib.pvsystem.sapm_effective_irradiance(
                    total_irradiance['poa_direct'], total_irradiance['poa_diffuse'],
                    am_abs, aoi, system['module'])
                dc = pvlib.pvsystem.sapm(effective_irradiance, temps, system['module'])
                ac = pvlib.pvsystem.snlinverter(dc['v_mp'], dc['p_mp'], system['inverter'])
                annual_energy = ac.sum()
                annual_irradiance = total_irradiance['poa_global'].sum()

                for panel_count in system['panel_count']:
                    print('panel count:', str(panel_count), 'azimuth:', str(azim), 'tilt', str(tilt))
                    deficiency = (loadProfile * 1000 - (ac * panel_count))
                    annual_deficiency = sum(deficiency[deficiency > 0])
                    annual_surplus = sum(deficiency[deficiency < 0])  # TODO:newhere
                    temp_final_panel_count.append(panel_count)
                    temp_final_azimuth.append(azim)
                    temp_final_tilt.append(tilt)
                    temp_final_annual_total_irradiance.append(annual_irradiance)
                    temp_final_annual_total_AC.append(annual_energy / 1000)
                    temp_final_annual_balance_kWh_deficit.append(annual_deficiency / 1000)
```

```
                    temp_final_annual_balance_kWh_surplus.append(annual_surplus / 1000)
                    temp_final_annual_ac_deficit_cost_EUR.append(
                        (annual_deficiency / 1000) * energy_import_price + 140 + (
                                (system['module']['Area'] * 269.09 * panel_count + 1319.4) / 25))
                    temp_final_annual_ac_balance_EUR.append((annual_deficiency / 1000 *
energy_import_price + 140 + (
                            (system['module']['Area'] * 269.09 * panel_count + 1319.4) / 25)) + (
                                                                    annual_surplus / 1000 *
energy_export_price))

    CostPerPositionAndQuantity = {'panel_count': temp_final_panel_count, 'azimuth':
temp_final_azimuth,
                                  'tilt': temp_final_tilt, 'annual_irradiance':
temp_final_annual_total_irradiance,
                                  'annual_ac': temp_final_annual_total_AC,
                                  'annual_balance_deficit': temp_final_annual_balance_kWh_deficit,
                                  'annual_balance_surplus': temp_final_annual_balance_kWh_surplus,
                                  'annual_ac_deficit_cost_EUR': temp_final_annual_ac_deficit_cost_EUR,
                                  }
    print(cost_function)

    global PVnominalPower
    PVnominalPower = system['module']['Vmpo'] * system['module']['Impo']
    if cost_function == 'cost_function_1':

        results = pd.DataFrame.from_dict(CostPerPositionAndQuantity,
orient='index').round(1).transpose()
        results['annual_ac_balance_EUR'] = (results['annual_balance_deficit'] * energy_import_price +
140 + (
                (system['module']['Area'] * 269.09 * results['panel_count'] + 1319.4) / PV_lifespan))
+ (
                                                (results['annual_balance_surplus']) *
energy_export_price)
        results['balance_EUR_per_m2'] = results['annual_ac_balance_EUR'] / (
                results['panel_count'] * system['module']['Area'])
        results['deficit_cost_EUR_per_m2'] = results['annual_ac_deficit_cost_EUR'] / (
                results['panel_count'] * system['module']['Area'])
        results['system_cost_per_m2'] = (269.09 * results['panel_count'] * system['module']['Area'] +
1319.4) / (
                results['panel_count'] * system['module']['Area'])

    elif cost_function == 'cost_function_2':
        results = pd.DataFrame.from_dict(CostPerPositionAndQuantity,
orient='index').round(1).transpose()

        results['annual_ac_balance_EUR'] = (results['annual_balance_deficit'] * energy_import_price +
140 + (
                (system['module']['Area'] * 269.09 * results['panel_count'] + 1319.4 + (
                        PVnominalPower * results['panel_count'] * inverter_price_index[
                    inverter_price_index_type])) / PV_lifespan)) + (
                                                results['annual_balance_surplus'] *
energy_export_price)

        results['balance_EUR_per_m2'] = results['annual_ac_balance_EUR'] / (
                results['panel_count'] * system['module']['Area'])
        results['deficit_cost_EUR_per_m2'] = results['annual_ac_deficit_cost_EUR'] / (
                results['panel_count'] * system['module']['Area'])
        results['system_cost_per_m2'] = (269.09 * results['panel_count'] * system['module']['Area'] +
1319.4) / (
                results['panel_count'] * system['module']['Area'])

    elif cost_function == 'cost_function_3':

        module_price_index = {'high_efficiency': 0.32, 'maistream': 0.24, 'low_cost': 0.16}  # july
2020 exc.vat
        results = pd.DataFrame.from_dict(CostPerPositionAndQuantity,
orient='index').round(1).transpose()
        results['balance_EUR_per_m2'] = results['annual_ac_balance_EUR'] / (
                results['panel_count'] * system['module']['Area'])
        results['deficit_cost_EUR_per_m2'] = results['annual_ac_deficit_cost_EUR'] / (
                results['panel_count'] * system['module']['Area'])
        results['system_cost_per_m2'] = (269.09 * results['panel_count'] * system['module']['Area'] +
1319.4) / (
                results['panel_count'] * system['module']['Area'])

    fig = px.line(results, x="panel_count",
                  y="annual_ac_balance_EUR",
                  animation_frame="tilt",
                  range_x=[5, 50],  # range_y=[2100,2600],
                  hover_data=['azimuth', 'azimuth'],
                  line_group="azimuth",
                  color='azimuth',
                  height=550,
                  title=" Energy cost balance "

                  ).update_traces(mode='lines+markers')
```

```
    # fig.data[0].update(mode='markers+lines')
    fig["layout"].pop("updatemenus")
    # fig.show()
    return fig


# TEST the above function
# generate_cost_plot('mainstream','Canadian_Solar_CS5P_220M___2009_',
'ABB__MICRO_0_25_I_OUTD_US_208_208V__CEC_2014_',0.07,0.15,'cost_function_1', dt(2019, 1, 1), dt(2019,
12, 31))

def update_cost_plot(inverter_price_index_type, energy_export_price, energy_import_price,
cost_function,
                     investment_cycle):
    PV_lifespan = investment_cycle  # TODO:fix naming
    if cost_function == 'cost_function_1':
        results['annual_ac_balance_EUR'] = (results['annual_balance_deficit'] * energy_import_price +
140 + (
                (system['module']['Area'] * 269.09 * results['panel_count'] + 1319.4) / PV_lifespan))
+ (
                                                results['annual_balance_surplus'] *
energy_export_price)

    elif cost_function == 'cost_function_2':
        results['annual_ac_balance_EUR'] = (results['annual_balance_deficit'] * energy_import_price +
140 + (
                (system['module']['Area'] * 269.09 * results['panel_count'] + 1319.4 + (
                        PVnominalPower * results['panel_count'] * inverter_price_index[
                    inverter_price_index_type])) / PV_lifespan)) + (
                                                results['annual_balance_surplus'] *
energy_export_price)

    elif cost_function == 'cost_function_3':
        results['annual_ac_balance_EUR'] = (results['annual_balance_deficit'] * energy_import_price +
140 + (
                (system['module']['Area'] * 269.09 * results['panel_count'] + 1319.4 + (
                        PVnominalPower * panel_count * inverter_price_index[
                    inverter_price_index_type])) / PV_lifespan)) + (
                                                results['annual_balance_surplus'] *
energy_export_price)
    fig = px.line(results, x="panel_count",
                  y="annual_ac_balance_EUR",
                  animation_frame="tilt",
                  range_x=[5, 50],  # range_y=[2100,2600],
                  hover_data=['azimuth', 'azimuth'],
                  line_group="azimuth",
                  color='azimuth',
                  height=550,

                  ).update_traces(mode='lines+markers')
    # fig.data[0].update(mode='markers+lines')
    fig["layout"].pop("updatemenus")
    # fig.show()
    return fig


def generate_contour_plot(dd_select_module, dd_select_inverter, panel_count, base_load, output_type,
start, end,
                          weather_file, electricity_use_profile_file, electricity_profile_element,
energy_export_price,
                          energy_import_price):
    energies = {}
    profiles = {}
    ACperOrient = {}
    AC_defficiency = {}
    AC_surplus = {}
    AC_defficiency_cost_EUR = {}
    AC_annual_balance_EUR = {}
    baseLoad = base_load

    naive_times = pd.date_range(start=start, end=end, freq='1h')
    system = {'module': PV_modules_list[dd_select_module], 'inverter':
sapm_inverters_list[dd_select_inverter],
              'surface_azimuth': surface_azimuths, 'surface_tilt': surface_tilts}

    # prepare epw
    epw_file_path = '/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files/' +
weather_file
    epw_data = pvlib.iotools.read_epw(epw_file_path, coerce_year=2019)
    # extract needed data from epw
    temp_air = epw_data[0]['temp_air']
    wind_speed = epw_data[0]['wind_speed']

    coordinates = [(epw_data[1]['latitude'], epw_data[1]['longitude'], epw_data[1]['city'],
epw_data[1]['altitude'],
                    epw_data[1]['TZ'])]
```

```
    # prepare energy use profile
    # Energy Usage profiles
    if output_type == "deficiency_prof" or output_type == "cost_balance":
        AnnnualHourlyEnergyUseProfileFile =
'/Users/dw/PycharmProjects/37project/Ontology/PV_panels/Energy_Use_Profiles/' +
electricity_use_profile_file
        eleConsumptionProfile = pd.read_csv(AnnnualHourlyEnergyUseProfileFile,
                                            index_col='Date/Time', parse_dates=True)
        eleConsumptionProfile[electricity_profile_element].groupby(
            eleConsumptionProfile[electricity_profile_element].index.hour).sum()
        loadProfile = eleConsumptionProfile[electricity_profile_element]
        print("Annual load: " + str(int(loadProfile.sum()) / 1000) + '[kWh]')
        loadProfile = loadProfile.tz_localize(int(epw_data[1]['TZ']) * 60 * 60)

    for latitude, longitude, name, altitude, timezone in coordinates:
        times = naive_times.tz_localize(int(timezone) * 60 * 60)  # localizes to GMT
        solpos = pvlib.solarposition.get_solarposition(times, latitude, longitude)
        dni_extra = pvlib.irradiance.get_extra_radiation(times)
        airmass = pvlib.atmosphere.get_relative_airmass(solpos['apparent_zenith'])
        pressure = pvlib.atmosphere.alt2pres(altitude)
        am_abs = pvlib.atmosphere.get_absolute_airmass(airmass, pressure)
        tl = pvlib.clearsky.lookup_linke_turbidity(times, latitude, longitude)
        cs = pvlib.clearsky.ineichen(solpos['apparent_zenith'], am_abs, tl, dni_extra=dni_extra,
altitude=altitude)
        for azim in system['surface_azimuth']:
            temp_irr_val = []  # to get all annual values from a single azimuth
            temp_AC_val = []  # to get all annual AC output per selected panel values from a single
azimuth
            temp_deficiency = []
            temp_surplus = []
            temp_final_annual_ac_deficit_cost_EUR = []
            temp_final_annual_ac_balance_EUR = []
            for tilt in surface_tilts:
                # print('tilt:',str(tilt),'Azimuth:',str(azim))
                aoi = pvlib.irradiance.aoi(tilt, azim, solpos['apparent_zenith'], solpos['azimuth'])
                total_irrad = pvlib.irradiance.get_total_irradiance(tilt,
                                                                    azim,
                                                                    solpos['apparent_zenith'],
                                                                    solpos['azimuth'],
                                                                    epw_data[0]['dni'],
epw_data[0]['ghi'],
                                                                    epw_data[0]['dhi'],
                                                                    dni_extra=dni_extra,
                                                                    model=irradiance_model)
                temps =
pvlib.pvsystem.temperature.sapm_cell(total_irrad['poa_global'].tz_convert(timezone),
                                                                  epw_data[0]['temp_air'],
epw_data[0]['wind_speed'],
                                                                  **params)

                effective_irradiance = pvlib.pvsystem.sapm_effective_irradiance(
                    total_irrad['poa_direct'], total_irrad['poa_diffuse'],
                    am_abs, aoi, system['module'])
                dc = pvlib.pvsystem.sapm(effective_irradiance, temps, system['module'])
                ac = pvlib.pvsystem.snlinverter(dc['v_mp'], dc['p_mp'], system['inverter'])
                # sum(ac[ac>100])
                # deficiency = (baseLoad - (ac*panelsQuantity))#in Wh

                if output_type == 'deficiency_prof' or output_type == "cost_balance":
                    deficiency = (loadProfile * 1000 - (ac * panel_count))

                elif output_type == "deficiency_const":  # todo maybe fix it for execution time???
                    deficiency = (baseLoad - (ac * panel_count))

                annual_energy = ac.sum()
                temp_AC_val.append(annual_energy * panel_count / 1000)  # to get kWh
                annual_irradiance = total_irrad['poa_global'].sum()
                temp_irr_val.append(annual_irradiance * panel_count / 1000)
                if output_type == 'deficiency_prof' or output_type == "deficiency_const" or
output_type == "cost_balance":
                    annual_deficiency = sum(deficiency[deficiency > 0])  # todo: by putting this under
if and so on
                    temp_deficiency.append(annual_deficiency / 1000)
                if output_type == "cost_balance":
                    annual_surplus = sum(deficiency[deficiency < 0])  # todo: by putting this under if
and so on
                    temp_surplus.append(annual_surplus / 1000)
                    temp_final_annual_ac_deficit_cost_EUR.append(
                        (annual_deficiency / 1000) * energy_import_price + 140 + (
                                (system['module']['Area'] * 269.09 * panel_count + 1319.4) / 25))
                    temp_final_annual_ac_balance_EUR.append((annual_deficiency / 1000 *
energy_import_price + 140 + (
                            (system['module']['Area'] * 269.09 * panel_count + 1319.4) / 25)) + (
                                                                    annual_surplus / 1000 *
energy_export_price))
```

```
                profiles[str(azim) + '-' + str(tilt)] = total_irrad['poa_global'].groupby(
                    total_irrad['poa_global'].index.hour).sum()

            energies[azim] = temp_irr_val
            ACperOrient[azim] = temp_AC_val
            if output_type == 'deficiency_prof' or output_type == "deficiency_const" or output_type ==
"cost_balance":
                AC_defficiency[azim] = temp_deficiency

            if output_type == "cost_balance":
                AC_surplus[azim] = temp_surplus
                AC_defficiency_cost_EUR[azim] = temp_final_annual_ac_deficit_cost_EUR
                AC_annual_balance_EUR[azim] = temp_final_annual_ac_balance_EUR
            # hour based irradiance profile

    # Make nice dateframe matrices:

    print(output_type)

    if output_type == 'total_Irr':
        plotTitle = 'Total Irradiance [kWh/m2] per orientation. Location: ' + str(coordinates[0][2]) +
' Lat: ' + str(
            coordinates[0][0]) + ' Lon:' + str(coordinates[0][1])
        plotInput = pd.DataFrame.from_dict(energies, orient='index').round(1).transpose()  # Wh/m2
        plotInput.index = surface_tilts  # kWh/m2

    elif output_type == 'total_AC':
        print('here' + output_type)
        print(system['module'])
        plotTitle = 'Total AC per orientation. ' + str(
            panel_count) + ' Panels. Area: ' + str(system['module']['Area'].__round__(2) *
panel_count) + 'm2. ' + str(
            coordinates[0][2]) + ' Lat: ' + str(coordinates[0][0]) + ' Lon:' + str(coordinates[0][1])
        plotInput = pd.DataFrame.from_dict(ACperOrient, orient='index').round(1).transpose()  # Wh/m2
        plotInput.index = surface_tilts  # kWh/m2

    elif output_type == 'deficiency_prof':
        print('selected')
        plotTitle = 'Deficiency per panel orientation. ' + str(
            panel_count) + ' Panels. Area: ' + str(
            round(system['module']['Area'] * panel_count)) + 'm2. ' + 'Panel Power rating: ' +
str(int(round(
            system['module']['Impo'] * system['module'][
                'Vmpo']))) + 'W ,' + ' Profile source:  ' + 'USA_TX_Austin.722540_TMY2.csv'
        plotInput = pd.DataFrame.from_dict(AC_defficiency, orient='index').round(1).transpose()  #
Wh/m2
        plotInput.index = surface_tilts  # kWh/m2
    elif output_type == 'deficiency_const':
        plotTitle = 'Deficiency per panel orientation. ' + str(
            panel_count) + ' Panels. Area: ' + str(
            system['module']['Area'].__round__(2) * panel_count) + 'm2. ' + ' Base load threshold: ' +
str(
            baseLoad / 1000) + 'kWh'
        plotInput = pd.DataFrame.from_dict(AC_defficiency, orient='index').round(1).transpose()  #
Wh/m2
        plotInput.index = surface_tilts  # kWh/m2
    elif output_type == "cost_balance":
        plotTitle = 'Annual cost balance (system,import,export). ' + str(
            panel_count) + ' Panels (' + str(
            system['module']['Area'].__round__(2) * panel_count) + 'm2). Investment cycle: 25 years.'
        plotInput = pd.DataFrame.from_dict(AC_annual_balance_EUR, orient='index').round(1).transpose()
# Wh/m2
        plotInput.index = surface_tilts  # kWh/m2
    else:
        plotTitle = '???'

    # plotTitle = 'Deficiency per panel orientation. ' + str(panelsQuantity) + ' Panels.' + ' Profile
source:  ' + 'USA_TX_Austin.722540_TMY2.csv'

    fig = go.Figure(data=go.Contour(
        z=plotInput,
        x=plotInput.columns,
        y=plotInput.index,
        hovertemplate=
        '<i>Azimuth</i>: %{x}°<br>' +
        '<i>Tilt</i>: %{y}°<br>' +
        '<b>kWh defic.</b>: %{z}</br>',
        name='',
        # colorscale='Electric',

        contours=dict(
            # coloring ='heatmap', smooths out the map
            showlabels=True,  # show labels on contours
            # start=plotInput.values.min(),
            # end=plotInput.values.max(),
```

```
            # size=50,#contour step
            labelfont=dict(  # label font properties
                size=12,
                color='white',

            ),
            # if we'd like to customize the bins
            # start=0,
            # end=8,
            # size=2,
            # dx=10,
            # x0=5,
            # dy=10,
            # y0=10,
        ),
        colorbar=dict(
            title='[kWh]',  # title here
            titleside='top',
            titlefont=dict(
                size=14,
                family="Courier New, monospace",
                color="#7f7f7f")
        )
    ))
    if output_type == 'total_AC' or output_type == 'total_Irr':
        fig.add_scatter(x=[plotInput.stack().idxmax()[1]], y=[plotInput.stack().idxmax()[0]],
                        mode="markers",
                        marker=dict(size=20, color="MediumPurple"),
                        name='',
                        hovertemplate=
                        '<i>Azimuth</i>: %{x}°<br>' +
                        '<i>Tilt</i>: %{y}°<br>' +
                        '<i>kWh</i>: ' + str(plotInput.values.max()) + '<br>',
                        # hoverinfo='none'
                        )
    elif output_type == "cost_balance":
        fig.add_scatter(x=[plotInput.stack().idxmin()[1]], y=[plotInput.stack().idxmin()[0]],
                        mode="markers",
                        marker=dict(size=20, color="MediumPurple"),
                        name='',
                        hovertemplate=
                        '<i>Azimuth</i>: %{x}°<br>' +
                        '<i>Tilt</i>: %{y}°<br>' +
                        '<i>EUR/year</i>: ' + str(plotInput.values.min()) + '<br>',
                        # hoverinfo='none'
                        )



    else:
        fig.add_scatter(x=[plotInput.stack().idxmin()[1]], y=[plotInput.stack().idxmin()[0]],
                        mode="markers",
                        marker=dict(size=20, color="MediumPurple"),
                        name='',
                        hovertemplate=
                        '<i>Azimuth</i>: %{x}°<br>' +
                        '<i>Tilt</i>: %{y}°<br>' +
                        '<i>kWh defic.</i>: ' + str(plotInput.values.min()) + '<br>',
                        # hoverinfo='none'
                        )

    if output_type == 'total_Irr':
        fig.data[0].colorbar.title = "[kWh/m2]"
    elif output_type == "cost_balance":
        fig.data[0].colorbar.title = "[EUR]"

    fig.update_layout(
        xaxis=dict(
            tickmode='linear',
            tick0=90,
            dtick=15
        ),
        yaxis=dict(
            tickmode='linear',
            tick0=0,
            dtick=5
        ),
        title=dict(text=plotTitle,
                   x=0.5,
                   y=1,
                   xanchor='center',
                   yanchor='top'
                   ),
        xaxis_title="Azimuth [deg]",
        yaxis_title="Tilt [deg]",
        font=dict(
```

```
            family="Courier New, monospace",
            size=12,
            color="#666666"
        ),
        height=550,  # try solving it with css
        margin=dict(l=20, r=20, t=35, b=20)
    )

    # fig.show()

    return fig


def visualize_energy_use_profile(electricity_use_profile_file,
                                 electricity_profile_element):  # TODO use source resd from column
names - list all with kWh
    # Create traces

    AnnnualHourlyEnergyUseProfileFile =
'/Users/dw/PycharmProjects/37project/Ontology/PV_panels/Energy_Use_Profiles/' +
electricity_use_profile_file
    eleConsumptionProfile = pd.read_csv(AnnnualHourlyEnergyUseProfileFile,
                                        index_col='Date/Time', parse_dates=True)
    # eleConsumptionProfile[electricity_profile_element].groupby(
    #    eleConsumptionProfile[electricity_profile_element].index.hour).sum()
    loadProfile = eleConsumptionProfile[electricity_profile_element]
    loadProfile = loadProfile.sort_index()

    fig = go.Figure()
    fig.add_trace(go.Scatter(x=loadProfile.index, y=loadProfile.values,
                             mode='lines',
                             name='Profile'))
    # fig.add_trace(go.Scatter(x=ac_converted.index, y=ac_converted.values,
    #                          mode='lines',
    #                          name='AC'))
    fig.update_layout(title_text='Electricity use profile; Source file: ' + str(
        electricity_use_profile_file) + ' Electricity use source: ' +
str(electricity_profile_element),
                      xaxis_rangeslider_visible=True)

    fig.show()


######################
# APPLICATION LAYOUT #
######################

app.layout = html.Div(
    className="container scalable",
    children=[
        html.Div(
            id="banner",
            className="banner",
            children=[
                html.H6("_PV Tool_"),
                # html.Img(src=app.get_asset_url("plotly_logo.png")),
            ],
        ),
        html.Div(
            className="app_main_content",
            children=[
                html.Div(
                    id="dropdown-select-outer",
                    # id="dropdown-select-1stRow",
                    children=[
                        html.Div(
                            [
                                html.P("Solar Module"),
                                dcc.Dropdown(
                                    id="dropdown-select-module",
                                    options=[{'label': i.replace("_", " "), 'value': i} for i in
                                             PV_modules_list.columns.sort_values()],
                                    # modules selector
                                    # options=[
                                    #     {"label": "Departure", "value": "dep"},
                                    #     {"label": "Arrival", "value": "arr"},
                                    # ],
                                    value='Canadian_Solar_CS5P_220M___2009_',
                                ),
                            ],
                            className="selector",
                        ),

                        html.Div([
                            html.Div(
                                [
```

```
                                    html.P(["Panel count"], style={"color": "#999999"}),
                                    dcc.Input(
                                        id="input-panel-count".format("number"),
                                        type="number",
                                        placeholder="10".format("number"),
                                        value=10,
                                    ),
                                ],
                                # className="selector",
                                style={"margin-right": "20px"},
                            ),

                        ],
                            className="selectormiddle",
                            style={'width': '17%'}

                        ),
                        html.Div(
                            [
                                html.P("Solar radiation data source:"),
                                dcc.Dropdown(
                                    id="dropdown-select-weather-file",
                                    options=[{'label': i.replace("_", " "), 'value': i} for i in
                                             weather_data_sources],
                                    value=weather_data_sources[0],

                                ),
                                # TODO initial value fix#html.P('Panels nominal Power Rating:
'+str(system['module']['Impo'] * system['module']['Vmpo'])+'W', style={ "color" : "#000000", "margin-
bottom" : "0", "font-size" : "0.7em", "line-height" : "1" }),

                            ],
                            id="weather-file-select",
                            className="selector",
                        ),

                    ],
                ),
                html.Div(
                    id="dropdown-select-2ndRow",
                    children=[
                        html.Div(
                            [
                                html.P("Inverter"),
                                dcc.Dropdown(
                                    id="dropdown-select-inverter",
                                    options=[{'label': i.replace("_", " "), 'value': i} for i in
                                             sapm_inverters_list.columns.sort_values()],
                                    value='ABB__MICRO_0_25_I_OUTD_US_208_208V__CEC_2014_',
                                ),
                            ],
                            className="selector",
                        ),

                        html.Div(
                            [
                                html.P("Date Range"),
                                dcc.DatePickerRange(
                                    id="date-picker-range",
                                    min_date_allowed=dt(2019, 1, 1),
                                    max_date_allowed=dt(2020, 1, 1),  # set maximum limit according to
local casting
                                    initial_visible_month=dt(2019, 1, 1),
                                    minimum_nights=1,
                                    display_format="MMM Do, YY",
                                    start_date=dt(2019, 1, 1),
                                    end_date=dt(2019, 12, 31),
                                ),
                            ],
                            id="date-picker-outer",
                            className="selectormiddle",
                            # style= {'width': 'auto'}
                            style={'width': '17%'}
                        ),

                        html.Div(
                            [
                                html.P("Select Output"),
                                dcc.Dropdown(
                                    id="dropdown-select-output",
                                    options=[
                                        {"label": "Total Irradiance", "value": "total_Irr"},
                                        {"label": "Total AC", "value": "total_AC"},
                                        {"label": "Ene. Deficiency b.o. Profile", "value":
"deficiency_prof"},
                                        {"label": "Ene. Deficiency b.o. Constant", "value":
```

```
"deficiency_const"},
                                        {"label": "Annual cost balance b.o. Profile", "value":
"cost_balance"},
                                    ],
                                    placeholder='Select desired output',
                                    # value='deficiency_prof',
                                ),
                                html.Div([
                                    html.P([""],
                                           id="profile-source-txt"),  # "Energy consumption profile
source"
                                    dcc.Dropdown(
                                        id="dropdown-select-electricity-use-profile",
                                        options=[{'label': i.replace("_", " "), 'value': i} for i in
                                                 eleConsumptionProfiles],
                                        placeholder="Select energy use profile source:",
                                    ),
                                    dcc.Dropdown(
                                        id="dropdown-select-profile-element",
                                        options=[
                                            {"label": "Electricity:Facility",
                                             "value": "Electricity:Facility [kWh](Hourly)"},
                                            {"label": "Heating:Electricity",
                                             "value": "Heating:Electricity [kWh](Hourly)"},
                                            {"label": "Cooling:Electricity",
                                             "value": "Cooling:Electricity [kWh](Hourly)"},
                                            {"label": "HVACFan:Fans:Electricity",
                                             "value": "HVACFan:Fans:Electricity [kWh](Hourly)"},
                                            {"label": "Electricity:HVAC", "value": "Electricity:HVAC
[kWh](Hourly)"},
                                            {"label": "Fans:Electricity", "value": "Fans:Electricity
[kWh](Hourly)"},
                                            {"label": "General:InteriorLights:Electricity",
                                             "value": "General:InteriorLights:Electricity
[kWh](Hourly)"},
                                            {"label": "Appl:InteriorEquipment:Electricity",
                                             "value": "Appl:InteriorEquipment:Electricity
[kWh](Hourly)"},
                                            {"label": "Misc:InteriorEquipment:Electricity",
                                             "value": "Misc:InteriorEquipment:Electricity
[kWh](Hourly)"},
                                            {"label": "Water Heater:WaterSystems:Electricity",
                                             "value": "Water Heater:WaterSystems:Electricity
[kWh](Hourly)"},

                                        ],
                                        placeholder='Select electricity consumption source',

                                    ),
                                ],
                                    id="dropdown-select-electricity-use-profile-wrapper"),

                                html.Div(

                                    [
                                        html.P("Base load [Wh]"),
                                        dcc.Input(
                                            id="input-base-load".format("number"),
                                            type="number",
                                            placeholder="1000".format("number"),
                                            value=1000,
                                        ),

                                    ],
                                    id="base-load-visibility-div",
                                    style={"margin-right": "20px"},
                                ),
                                # Options for annual cost balance visualization
                                html.Div([
                                    html.P([""], id="annual-cost-balance-txt"),
                                    html.Div(
                                        [
                                            html.P("Enenrgy import price(EUR)"),

                                            dcc.Input(
                                                id="input-energy-import-price-
contour".format("number"),

                                                type="number",
                                                min=0.00,
                                                step=0.01,
                                                max=1,
                                                placeholder="0.15".format("number"),
                                                value=0.15,
                                            ),
                                            html.P("Enenrgy export price(EUR)"),
                                            dcc.Input(
```

```
                                        id="input-energy-export-price-
contour".format("number"),
                                        type="number",
                                        min=0.00,
                                        step=0.01,
                                        max=1,
                                        placeholder="0.07".format("number"),
                                        value=0.07,
                                    ),

                                ],
                                # className="selector",
                                # style={"margin-top": "100px", "margin-right": "20px"},
                            ),

                            dcc.Dropdown(
                                id="dropdown-select-electricity-use-profile-contour",
                                options=[{'label': i.replace("_", " "), 'value': i} for i in
                                         eleConsumptionProfiles],
                                placeholder="Select energy use profile source:",
                            ),
                            dcc.Dropdown(
                                id="dropdown-select-profile-element-contour",
                                options=[
                                    {"label": "Electricity:Facility",
                                     "value": "Electricity:Facility [kWh](Hourly)"},
                                    {"label": "Heating:Electricity",
                                     "value": "Heating:Electricity [kWh](Hourly)"},
                                    {"label": "Cooling:Electricity",
                                     "value": "Cooling:Electricity [kWh](Hourly)"},
                                    {"label": "HVACFan:Fans:Electricity",
                                     "value": "HVACFan:Fans:Electricity [kWh](Hourly)"},
                                    {"label": "Electricity:HVAC", "value": "Electricity:HVAC
[kWh](Hourly)"},
                                    {"label": "Fans:Electricity", "value": "Fans:Electricity
[kWh](Hourly)"},
                                    {"label": "General:InteriorLights:Electricity",
                                     "value": "General:InteriorLights:Electricity
[kWh](Hourly)"},
                                    {"label": "Appl:InteriorEquipment:Electricity",
                                     "value": "Appl:InteriorEquipment:Electricity
[kWh](Hourly)"},
                                    {"label": "Misc:InteriorEquipment:Electricity",
                                     "value": "Misc:InteriorEquipment:Electricity
[kWh](Hourly)"},
                                    {"label": "Water Heater:WaterSystems:Electricity",
                                     "value": "Water Heater:WaterSystems:Electricity
[kWh](Hourly)"},

                                ],
                                placeholder='Select electricity consumption source',

                            ),
                        ],
                            id="annual-cost-balance-options-wrapper"),

                        html.Div(id='mockup-button'),
                        dcc.Input(style={'display': 'none'}),

                    ],
                    className="selector",

                ),

                html.Div(
                    [
                        html.P("_"),
                        html.Button(id='submit-button-state', n_clicks=0, children='Submit ',
                                    style={"backgroundColor": "greenyellow"}),
                        html.Div([
                            html.P("_ "),
                            html.Button(id='preview-profile-state', n_clicks=0,
children='Preview',
                                        style={"backgroundColor": "white"})
                        ],
                            id="prewiew-button-wrapper"),
                    ],

                ),
            ],
        ),
        # -------------------------
        # Contour plot div
        html.Div(
            id="top-row",
            className="row",
```

```
                    children=[
                        html.Div(
                            id="map_geo_outer",
                            className="twelve columns",
                            children=dcc.Loading(
                                # contour plot
                                children=dcc.Graph(id="contours")
                            ),

                        ),

                    ],
                    style={"width": "100%"},

                ),
                # ------------------------
                html.Div(
                    id="dropdown-select-3rdRow",
                    children=[
                        html.Div(
                            [
                                html.P("Enenrgy import price(EUR)"),
                                dcc.Input(
                                    id="input-energy-import-price".format("number"),
                                    type="number",
                                    min=0.00,
                                    step=0.01,
                                    max=1,
                                    placeholder="0.15".format("number"),
                                    value=0.15,
                                ),

                            ],
                            # className="selector",
                            style={"margin-top": "100px", "margin-right": "20px"},
                        ),
                        html.Div(
                            [
                                html.P("Enenrgy export price(EUR)"),
                                dcc.Input(
                                    id="input-energy-export-price".format("number"),
                                    type="number",
                                    min=0.00,
                                    step=0.01,
                                    max=1,
                                    placeholder="0.07".format("number"),
                                    value=0.07,
                                ),

                            ],
                            # className="selector",
                            style={"margin-top": "100px", "margin-right": "20px"},
                        ),
                        html.Div(
                            [
                                html.P("Select Cost Function"),
                                dcc.Dropdown(
                                    id="dropdown-select-cost-function",
                                    options=[
                                        {"label": "CF (Schrack Technik)", "value": "cost_function_1"},
                                        {"label": "CF (Österreichische Energieagentur)", "value":
"cost_function_2"},
                                        # {"label": "Cost Function 3", "value": "cost_function_3"},
                                    ],
                                    placeholder='Cost Function 1',
                                    value='cost_function_1',
                                ),
                                html.Div(
                                    [
                                        html.P("Select inverter class"),
                                        dcc.Dropdown(
                                            id="dropdown-select-inverter-type",
                                            options=[
                                                {"label": "High end", "value": "high_end"},
                                                {"label": "Mainstream", "value": "mainstream"},
                                                {"label": "Low cost", "value": "low_cost"},
                                            ],
                                            placeholder='Mainstream',
                                            value='mainstream',
                                        ),
                                    ],
                                    id="inverter-class-wrapper"
                                ),

                            ],
                            className="selector",
```

```
                            style={"margin-top": "100px", "margin-right": "20px"},
                        ),

                        html.Div(
                            [
                                html.P("Inv. cycle(y)"),
                                dcc.Input(
                                    id="input-investment-cycle".format("number"),
                                    type="number",
                                    min=5,
                                    step=1,
                                    max=25,
                                    placeholder="25".format("number"),
                                    value=25,

                                ),
                            ],
                            style={"margin-top": "100px", "margin-right": "20px"},

                        ),

                        html.Div(
                            [
                                html.P("_______________"),
                                html.Button(id='submit-button-state-cost', n_clicks=0,
children='Submit'),
                            ],
                            style={"margin-top": "100px"},

                        ),
                    ],
                ),
                html.Div(id='date-text-output'),

                # 2nd plot (costs )div
                html.Div(
                    id="bottom-row",
                    className="row",
                    children=[
                        html.Div(
                            id="cost_bottom",
                            className="twelve columns",
                            children=dcc.Loading(
                                # avg arrival/dep delay by destination state
                                children=dcc.Graph(id="cost_line")
                            ),
                        ),

                    ],
                    style={"width": "100%", },

                ),

            ],
        ),
    ],
)


# Print date before 2nd plot
@app.callback(Output('date-text-output', 'children'),
              [Input("date-picker-range", "start_date"),
               Input("date-picker-range", "end_date"),
               Input("dropdown-select-weather-file", "value"),
               Input("dropdown-select-electricity-use-profile", "value"),
               Input("dropdown-select-profile-element", "value"),
               Input("input-investment-cycle", "value")
               ])
def update(start, end, solar_source, energy_use_profile, profile_element, investment_cycle):
    print(type(start))
    # return f'Time range: {start.replace("T00:00:00", " ")} - {end.replace("T00:00:00", " ")}'
    return f'INPUT DETAILS  -  Time range: {start.replace("T00:00:00", " ")} -
{end.replace("T00:00:00",
    " ")}.Investment
    cycle: {investment_cycle}
    ' \
           f'
    years.Solar
    radiation
    data
    src.: {solar_source}.Energy
    use
    profile
    src.: {energy_use_profile}.Energy
    use
    src.: {profile_element}.
```

```
    '


def generate_mockup_fig():
    #    df = px.data.gapminder().query("country=='Canada'")
    #    fig = px.line(df, x="year", y="lifeExp", title='Life expectancy in Canada')
    fig = go.Figure()
    fig.update_layout(
        xaxis={"visible": False},
        yaxis={"visible": False},
        annotations=[
            dict(
                xref="paper",
                yref="paper",
                text="SUBMIT INPUT SETTINGS",
                showarrow=False,
                font=dict(size=28)

            )
        ]

    )

    return fig


@app.callback(
    Output("contours", "figure"),
    [Input('submit-button-state', 'n_clicks')],
    [
        State("dropdown-select-module", "value"),
        State("dropdown-select-inverter", "value"),
        State("input-panel-count", "value"),
        State("input-base-load", "value"),
        State("dropdown-select-output", "value"),
        State("date-picker-range", "start_date"),
        State("date-picker-range", "end_date"),
        State("dropdown-select-weather-file", "value"),
        State("dropdown-select-electricity-use-profile-contour", "value"),
        State("dropdown-select-profile-element-contour", "value"),
        State("input-energy-export-price-contour", "value"),
        State("input-energy-import-price-contour", "value"),
    ],

)
def update_contours(n_clicks, dd_select_module, dd_select_inverter, panel_count, base_load,
output_type, start, end,
                    weather_file, electricity_use_profile_file, electricity_profile_element,
energy_export_price,
                    energy_import_price):
    # Update contour when dropdown or date-picker change
    print(n_clicks)
    if n_clicks != 0:
        if dd_select_module is None:
            dd_select_module = 'Canadian_Solar_CS5P_220M___2009_'

        if dd_select_inverter is None:
            dd_select_inverter = 'ABB__MICRO_0_25_I_OUTD_US_208_208V__CEC_2014_'

        if panel_count is None:
            panel_count = 10

        if base_load is None:
            base_load = 1000

        if output_type is None:
            output_type = "total_Irr"

        if electricity_use_profile_file is None:
            electricity_use_profile_file = eleConsumptionProfiles[0]

        if electricity_profile_element is None:
            electricity_profile_element = "Electricity:Facility [kWh](Hourly)"

        start, end = start.replace("T", " "), end.replace("T", " ")
        return generate_contour_plot(dd_select_module, dd_select_inverter, panel_count, base_load,
output_type, start,
                                     end, weather_file, electricity_use_profile_file,
electricity_profile_element,
                                     energy_export_price,
                                     energy_import_price)
    else:
        return generate_mockup_fig()


@app.callback(
```

```
    Output("cost_line", "figure"),

    [Input('submit-button-state-cost', 'n_clicks')],
    [
        State("dropdown-select-inverter-type", "value"),
        State("dropdown-select-module", "value"),
        State("dropdown-select-inverter", "value"),
        State("input-energy-export-price", "value"),
        State("input-energy-import-price", "value"),
        State("dropdown-select-cost-function", "value"),
        State("date-picker-range", "start_date"),
        State("date-picker-range", "end_date"),
        State("dropdown-select-weather-file", "value"),
        State("dropdown-select-electricity-use-profile", "value"),
        State("dropdown-select-profile-element", "value"),
        State("input-investment-cycle", "value"),

    ],

)
def update_line_cost(n_clicks, inverter_price_index_type, dd_select_module, dd_select_inverter,
energy_export_price,
                     energy_import_price,
                     cost_function, start, end, weather_file, electricity_use_profile_file,
electricity_profile_element,
                     investment_cycle):
    # Update contour when dropdown or date-picker change
    print(n_clicks)
    if n_clicks == 1:
        if inverter_price_index_type is None:
            inverter_price_index_type = 'mainstream'

        if dd_select_module is None:
            dd_select_module = 'Canadian_Solar_CS5P_220M___2009_'

        if dd_select_inverter is None:
            dd_select_inverter = 'ABB__MICRO_0_25_I_OUTD_US_208_208V__CEC_2014_'

        if energy_export_price is None:
            energy_export_price = 0.07

        if energy_import_price is None:
            energy_import_price = 0.15

        if cost_function is None:
            cost_function = 1
        if investment_cycle is None:
            investment_cycle = 25

        # TODO: need to add title to cost function so that we know what#s on the graph
        if electricity_use_profile_file is None:
            electricity_use_profile_file = eleConsumptionProfiles[0]

        if electricity_profile_element is None:
            electricity_profile_element = "Electricity:Facility [kWh](Hourly)"

        start, end = start.replace("T", " "), end.replace("T", " ")

        print(n_clicks)

        return generate_cost_plot(inverter_price_index_type, dd_select_module, dd_select_inverter,
energy_export_price,
                                  energy_import_price,
                                  cost_function, start, end, weather_file,
electricity_use_profile_file,
                                  electricity_profile_element, investment_cycle)
    elif n_clicks > 1:

        return update_cost_plot(inverter_price_index_type, energy_export_price, energy_import_price,
cost_function,
                                investment_cycle)


    else:
        return generate_mockup_fig()


# reset clicks on
@app.callback(Output('submit-button-state-cost', 'n_clicks'),
              [Input("date-picker-range", "start_date"),
               Input("date-picker-range", "end_date"),
               Input("dropdown-select-profile-element", "value"),
               Input("dropdown-select-output", "value"),
               Input("dropdown-select-weather-file", "value")
               ])
def update(start, end, temp_val_1, temp_val_2, temp_val_3):
```

```
    return 0


# preview button energy profile cost plot
# reset clicks on
@app.callback(
    Output('mockup-button', 'children'),
    [Input('preview-profile-state', 'n_clicks'), ],
    [State("dropdown-select-electricity-use-profile", "value"),
     State("dropdown-select-profile-element", "value"),
     ])
def update(n_clicks, electricity_use_profile_file, electricity_profile_element):
    print("Preview btn click state: " + str(n_clicks))
    if n_clicks != 0:
        visualize_energy_use_profile(electricity_use_profile_file, electricity_profile_element)
    return ""  # return nothing


# reset preview butto
@app.callback(Output('preview-profile-state', 'n_clicks'),
              [Input("dropdown-select-electricity-use-profile", "value"),
               Input("dropdown-select-profile-element", "value"),
               Input("dropdown-select-output", "value"),
               ])
def update(empty_1, empty_2, empty_3):
    return 0


# inverter type dropdown visibility control
@app.callback(Output('inverter-class-wrapper', 'style'),
              [Input("dropdown-select-cost-function", "value"),
               ])
def update(dropdown_value):
    if dropdown_value == "cost_function_2":
        print(dropdown_value)
        return {'display': 'block'}
    else:
        return {'display': 'none'}


# control profile selection display
@app.callback(Output('dropdown-select-electricity-use-profile-wrapper', 'style'),
              [Input("dropdown-select-output", "value"),
               ])
def update(dropdown_value):
    if dropdown_value == "deficiency_prof":
        print(dropdown_value)
        return {'display': 'block'}
    else:
        return {'display': 'none'}


@app.callback(Output('profile-source-txt', 'style'),
              [Input("dropdown-select-output", "value"),
               ])
def update(dropdown_value):
    if dropdown_value == "deficiency_prof":
        print(dropdown_value)
        return {'display': 'block'}
    else:
        return {'display': 'none'}


@app.callback(Output('base-load-visibility-div', 'style'),
              [Input("dropdown-select-output", "value"),
               ])
def update(dropdown_value):
    if dropdown_value == "deficiency_const":
        print(dropdown_value)
        return {'display': 'block'}
    else:
        return {'display': 'none'}


@app.callback(Output('prewiew-button-wrapper', 'style'),
              [Input("dropdown-select-output", "value"),
               ])
def update(dropdown_value):
    if dropdown_value == "deficiency_prof":
        print(dropdown_value)
        return {'display': 'block'}
    else:
        return {'display': 'none'}


####
```

```python
@app.callback(Output("annual-cost-balance-options-wrapper", 'style'),
              [Input("dropdown-select-output", "value"),
               Input('submit-button-state-cost', 'n_clicks')
               ])
def update(dropdown_value, n_clicks):
    if dropdown_value == "cost_balance":
        print(dropdown_value)
        return {'display': 'block'}
    else:
        return {'display': 'none'}

    # Run the server


if __name__ == "__main__":
    app.run_server(
        debug=True, port=8054, dev_tools_hot_reload=False, use_reloader=False
    )
```

```
################################################
# BPD ontology to LADYBUG
# List unique attributes in the selected file v1.0
#
################################################

import scriptcontext
import ghpythonremote
h5py = scriptcontext.sticky['h5py']
np=  scriptcontext.sticky['numpy']
import Grasshopper.Kernel as gh

#=================
if _hdf5File != None and _attribute != None:

    hdf5File = _hdf5File

    objects_list=[]
    variables_name_list=[]
    uniqueNamesList=[]

    print "looking for:   ", _attribute,
    #print len(_objects_list)
    #print _hdf5File


    def findspecial(name, obj):
        #print type(obj)
        #print name
        try:
            curName = ghpythonremote.obtain(obj.attrs.get(_attribute))
            #print curName
            if  curName != None:
                uniqueNamesList.append(curName)
        except:
            print "warning findspecial, it's OK though"



    while True:


        if _hdf5File != None and len(_attribute) != 0:
            #print "hdf5+"
            f = h5py.File(_hdf5File, 'r')
            try:
                ghpythonremote.obtain(f.visititems(findspecial))
            except:
                print "warning visititems, retrying"
                uniqueNamesList=[]
                continue
            print "done"
            uniqueNamesList = ghpythonremote.obtain(np.unique(uniqueNamesList).tolist())
            print uniqueNamesList
            break

        else:
            print "input missing"
            break
```

```
################################################
# BPD ontology to LADYBUG
# Filter variables based on selected attribute v4.2
#
################################################

import scriptcontext
import ghpythonremote

h5py = scriptcontext.sticky['h5py']
import Grasshopper.Kernel as gh

if run:

    objects_list = []
    variables_name_list = []
    print
    "looking for:   ", _attribute, " - ", _attributeValue

    def findspecial(name, obj):
        # print type(obj)
        # print name
        try:
            curName = obj.attrs.get(_attribute)
            # print curName
            if curName == _attributeValue:
                objects_list.append(obj)
                variables_name_list.append(name.split("/")[-1])
        except:
            print
            "warning findspecial, it's OK though"


    def findinobjlist(name, obj):
        # print obj.parent
        # print name
        try:
            curName = obj.attrs.get(_attribute)
            # print curName
            if curName == _attributeValue:
                objects_list.append(obj.parent)
                variables_name_list.append(obj.parent.name.split("/")[-1])
        except:
            print
            "warning findspecial, it's OK though"


    while True:

        if len(_objects_list) != 0 and _hdf5File is None:
            # print "obj+"
            # print _objects_list
            try:
                for x in _objects_list:
                    x.visititems(findinobjlist)
                    if x.attrs.get(_attribute) == _attributeValue:
                        objects_list.append(x)
                        variables_name_list.append(x.name.split("/")[-1])

            except:
                # print "warning visititems, retrying"
                objects_list = []
                variables_name_list = []
                break
                continue
            print
            "done_obj"
            break

        elif _hdf5File is not None and len(_objects_list) == 0:
            # print "hdf5+"
            f = h5py.File(_hdf5File, 'r')
            try:
                if _attribute == "Name" or _attribute == "Type" or _attribute == "Direction" or
_attribute == "Unit":
                    f.visititems(findspecial)
                else:
                    f.visititems(findinobjlist)

            except:
                print
                "warning visititems, retrying"
                objects_list = []
                variables_name_list = []
                continue
```

```
        print
        "done_hdf5"
        break

    elif _hdf5File is not None and len(_objects_list) != 0:

        warning = "As for now, only one input is allowed.\n" + \
                  "Either HDF5 File or HDF5 objects list.\n" + \
                  "Disconnect one of the sources and try again."
        w = gh.GH_RuntimeMessageLevel.Warning
        ghenv.Component.AddRuntimeMessage(w, warning)
        print
        "As for now, only one input is allowed."
        break
    else:
        print
        "no input"
        break
```

```
################################################
# BPD ontology to LADYBUG
# HDF5 object postprocessing Ladybug v1.5
# DateTime period, segmentation/aggregation,
# missing data handling
################################################

import sys
import scriptcontext
import ghpythonremote
import Grasshopper.Kernel as gh
import time
from datetime import datetime

h5py = scriptcontext.sticky['h5py']
np = scriptcontext.sticky['numpy']
pd = scriptcontext.sticky['pandas']


analysisPeriod = _analysisPeriod_


def mean(numbers):
    if type(numbers) == list:
        return float(sum(numbers)) / max(len(numbers), 1)
    else:
        return numbers


# =================
if _timeStep_ == None: _timeStep_ = 60;
step = int(_timeStep_) * 60
variable = _variable_hdf5_object

varTimestamps =
ghpythonremote.obtain(_variable_hdf5_object['TemporalDomain/TimeStamp'].value.tolist())
varDatapoints = ghpythonremote.obtain(_variable_hdf5_object['Magnitude'].value.tolist())

startDT = datetime(_analysisPeriod_[0][0], _analysisPeriod_[0][1], _analysisPeriod_[0][2],
                   _analysisPeriod_[0][3] - 1)
startTS = int(time.mktime(startDT.timetuple()))
stopDT = datetime(_analysisPeriod_[1][0], _analysisPeriod_[1][1], _analysisPeriod_[1][2],
_analysisPeriod_[1][3] - 1)
stopTS = int(time.mktime(stopDT.timetuple()))

if startTS < varTimestamps[0]:
    print
    "Data missing for this analysis period."
    print
    "First dataset entry timestamp is:"
    print
    datetime.fromtimestamp(varTimestamps[0]).strftime('%Y-%m-%d %H:%M:%S')
    errorMessage = "Data missing for this analysis period.\n" + \
                   "First dataset entry timestamp is:\n" + \
                   datetime.fromtimestamp(varTimestamps[0]).strftime('%Y-%m-%d %H:%M:%S')
    e = gh.GH_RuntimeMessageLevel.Error
    ghenv.Component.AddRuntimeMessage(e, errorMessage)

    sys.exit()

newTimestamp1 = []
newMagnitude1 = []
newTimestamp1 = []
finalValues = []
finalTimestamps = []
tempValue = []
i = 0

currentTimeband = startTS

# Averaging value based on the following time period from ztime stamp- 9:00 value is the average of
the datapoints  from 9:00-10:00
for idx, val in enumerate(varTimestamps):

    if currentTimeband > stopTS:  # stop when reached the endDate
        break
    elif val < currentTimeband:  # to discard entries lower than startDate
        continue
    elif val < currentTimeband + step:
        tempValue.append(varDatapoints[idx])

    else:
        while True:

            if len(tempValue) != 0 and val < currentTimeband + 2 * step:
                finalValues.append(round(mean(tempValue), 2))
```

```
                    finalTimestamps.append(currentTimeband)
                    tempValue = []
                    tempValue.append(varDatapoints[idx])
                    currentTimeband = currentTimeband + step
                    break
                elif len(tempValue) == 0 and val < currentTimeband + 2 * step:

                    finalValues.append(None)
                    finalTimestamps.append(currentTimeband)
                    tempValue.append(varDatapoints[idx])
                    currentTimeband = currentTimeband + step
                    break

                else:
                    if len(tempValue) != 0:
                        finalValues.append(round(mean(tempValue), 2))
                        finalTimestamps.append(currentTimeband)
                        tempValue = []
                    else:
                        finalValues.append(None)
                        finalTimestamps.append(currentTimeband)
                    currentTimeband = currentTimeband + step


    finalDateTime = []
    for x in finalTimestamps:
        finalDateTime.append(datetime.fromtimestamp(x).strftime('%Y-%m-%d %H:%M:%S'))

    if missingDataHandling_ != None and missingDataHandling_ != 'adjacent mean':  # a 'ffill' or 'bfill'
    or 'adjacent mean'):
        print
        "Interpolation method: ", missingDataHandling_
        missingData = pd.Series(finalValues)
        missingData = missingData.interpolate(method=missingDataHandling_)
        missingData = missingData.bfill().ffill()
        interpolatedFinalValues = ghpythonremote.obtain(missingData.tolist())

    elif missingDataHandling_ == 'adjacent mean':
        print
        "Interpolation method: ", 'adjacent mean inside'
        missingData = pd.Series(finalValues)
        missingData = (missingData.ffill() + missingData.bfill()) / 2
        missingData = missingData.bfill().ffill()
        interpolatedFinalValues = ghpythonremote.obtain(missingData.tolist())
```

```
################################################
# BPD ontology to LADYBUG
# Skyscanner BPD object to "selected sky matrix"#
# (diffused values only!!!)                     #
################################################


import sys
import scriptcontext
import ghpythonremote
import Grasshopper.Kernel as gh
import time
from datetime import datetime

start = time.time()
h5py = scriptcontext.sticky['h5py']
np = scriptcontext.sticky['numpy']
pd = scriptcontext.sticky['pandas']

print _analysisPeriod
print str(_analysisPeriod[0])

#=================
if startIndex == None: startIndex = 0;

#GENERATE DATETIME IN RANGE TO MATCH AVERAGES
startStr= str(_analysisPeriod[0][0]) + "-" + str(_analysisPeriod[0][1]) + "-"+
str(_analysisPeriod[0][2]) + " " +str(_analysisPeriod[0][3]-1) +":00:00"
endStr= str(_analysisPeriod[1][0]) + "-" + str(_analysisPeriod[1][1]) + "-"+
str(_analysisPeriod[1][2]) + " " +str(_analysisPeriod[1][3]-1) +":00:00"

print "From: "+startStr+" Till: "+ endStr

varTimestamps = pd.Series(_skyScannerObject['TemporalDomain/TimeStamp'][()])[int(startIndex):]
varDatapoints = pd.DataFrame(_skyScannerObject['Magnitude'][()],columns = range(1,
146))[int(startIndex):]

#!!!!!! WARNING !!!!!!
#OLDER HDF5 % IMPORTS WONT WORK CORRECTLY SINCE TIMESTAMP SAVED IN HDF5 IS BASED ON LOCAL POSIX TIME
AND NOT UTC
#THIS NEEDS TO BE FIXED IN THE HDF5 file
df1 = pd.DataFrame(pd.to_datetime(varTimestamps, unit='s'), columns=['datetime'])#Get all timestamps
and make datefreme from it
df1 = pd.concat([df1, varDatapoints], axis=1)#merge with datapoints
df1.set_index('datetime', inplace=True)

#Limit of analysis period:
df_p = df1.resample('H').mean()


#This is just a test, we have the Diffuse only but the whole set is needed (diff, glob, direct)
totalRad = ["key:location/dataType/units/frequency/startsAt/endsAt", "VIENNA_KARLSPLATZ_TU_WIEN", "Sky
Patches' Total Radiation", "kWh/m2", 'NA', (_analysisPeriod[0][1], _analysisPeriod[0][2],
_analysisPeriod[0][3]), (_analysisPeriod[1][1], _analysisPeriod[1][2], _analysisPeriod[1][3])]
totalRadValues = []

print "len: " , len(df_p.loc[startStr:endStr])

for each in (df_p.loc[startStr:endStr].sum()/1000).tolist():
    totalRadValues.append(each)

diffuseRad = ["key:location/dataType/units/frequency/startsAt/endsAt", "VIENNA_KARLSPLATZ_TU_WIEN",
"Sky Patches' Diffuse Radiation", "kWh/m2", 'NA', (_analysisPeriod[0][1], _analysisPeriod[0][2],
_analysisPeriod[0][3]), (_analysisPeriod[1][1], _analysisPeriod[1][2], _analysisPeriod[1][3])]
diffuseRadValues = []

for each in [0] * 145:
    diffuseRadValues.append(each)

directRad = ["key:location/dataType/units/frequency/startsAt/endsAt", "VIENNA_KARLSPLATZ_TU_WIEN",
"Sky Patches' Direct Radiation", "kWh/m2", 'NA', (_analysisPeriod[0][1], _analysisPeriod[0][2],
_analysisPeriod[0][3]), (_analysisPeriod[1][1], _analysisPeriod[1][2], _analysisPeriod[1][3])]
directRadValues = []
for each in [0] * 145:
    directRadValues.append(each)

#The data from the sky scanner is in W/m^2*sr and we need to normalize the value per patch area:
def calculatePatches(patchesValues):  # ret Sum if we want to get a cumulative sum

    horIllumPerPatch = []
    for x in patchesValues[0:30]:
        Ehp = x* 0.0435
        horIllumPerPatch.append(Ehp)

    for x in patchesValues[30:60]:
        Ehp = x* 0.0416
```

```
            horIllumPerPatch.append(Ehp)

        for x in patchesValues[60:84]:
            Ehp = x* 0.0474
            horIllumPerPatch.append(Ehp)

        for x in patchesValues[84:108]:
            Ehp = x* 0.0407
            horIllumPerPatch.append(Ehp)

        for x in patchesValues[108:126]:

            Ehp = x* 0.0429
            horIllumPerPatch.append(Ehp)

        for x in patchesValues[126:138]:
            Ehp = x* 0.0445
            horIllumPerPatch.append(Ehp)

        for x in patchesValues[138:144]:
            Ehp = x* 0.0455
            horIllumPerPatch.append(Ehp)

        Ehp = patchesValues[144]* 0.0344
        horIllumPerPatch.append(Ehp)
        return horIllumPerPatch

totalRadValues = calculatePatches(totalRadValues)

selectedSkyMtx = totalRad + totalRadValues + diffuseRad + diffuseRadValues + directRad +
directRadValues
selectedSkyMtx = ghpythonremote.obtain(selectedSkyMtx)
```

```
######################################################################
# PV energy generation/m2 per location and panel position strategy#
# v1.4                                                               #
######################################################################

import pvlib
from pvlib import location
from pvlib import irradiance
from pvlib import tracking
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

plt.rcParams['figure.figsize'] = (6, 4.5)
sns.set_color_codes()
sns.set_theme(style="whitegrid")

# Helsinki optimal Tilt:40
epw, metadata = pvlib.iotools.read_epw(

'/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files/FIN_Helsinki.029740_IWEC.epw
',
    coerce_year=2018)

# Vienna optimal Tilt: 30
# epw, metadata = pvlib.iotools.read_epw(
#
'/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files/AUT_Vienna.Schwechat.110360_
IWEC_official.epw',
#    coerce_year=2018)

# Santa Fe optimal Tilt:35
# epw, metadata =
pvlib.iotools.read_epw('/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files/USA_N
M_Santa.Fe.County.Muni.AP.723656_TMY3.epw', coerce_year=2018)

# Singapore optimal Tilt:0
# epw, metadata =
pvlib.iotools.read_epw('/Users/dw/PycharmProjects/37project/Ontology/PV_panels/EPW_Weather_files/SGP_S
ingapore.486980_IWEC.epw', coerce_year=2018)

# lat, lon, timezone from EPW file
location = location.Location.from_epw(metadata)

times = epw.index - pd.Timedelta('30min')
solar_position = location.get_solarposition(times)

solar_position.index += pd.Timedelta('30min')
solar_position.index = epw.index.tz_localize(None)

# get rid of ltime ocalization according to UTC
epw.index = epw.index.tz_localize(None)

# use apparent zenith as tilt
ManualTilt = solar_position['apparent_zenith'].copy()


def calculate_mean_tilt(positonChangePeroiod, apparentZenithArray, maxSurfaceTilt):
    if positonChangePeroiod == 'quarterly' or positonChangePeroiod == 1:
        period_mean_tilt = apparentZenithArray.copy()
        period_mean_tilt.loc[period_mean_tilt > maxSurfaceTilt] = np.NaN
        temp_dataframe = pd.DataFrame(index=solar_position.index)
        cosFunctArray = pd.Series(
            [np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN,
np.NaN, 1, np.NaN,
             np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN])  #
starts with 00:00
        temp_dataframe[''] = np.resize(cosFunctArray, temp_dataframe.shape[0])  # fill df with
repeating array
        cosNormalizedZenith = temp_dataframe[''] * period_mean_tilt
        periodNormalizedZenith = cosNormalizedZenith.resample('Q').mean()
        optimizedTiltAngles = apparentZenithArray.copy()

        for month in range(1, 4):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.month == month] =
periodNormalizedZenith[0]
        for month in range(4, 7):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.month == month] =
periodNormalizedZenith[1]
        for month in range(7, 10):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.month == month] =
periodNormalizedZenith[2]
        for month in range(10, 13):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.month == month] =
periodNormalizedZenith[3]
```

```
    if positonChangePeroiod == 'monthly' or positonChangePeroiod == 2:
        period_mean_tilt = apparentZenithArray.copy()
        period_mean_tilt.loc[period_mean_tilt > maxSurfaceTilt] = np.NaN
        temp_dataframe = pd.DataFrame(index=solar_position.index)
        cosFunctArray = pd.Series(
            [np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN,
np.NaN,
             1, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN,
             np.NaN])  # starts with 00:00
        temp_dataframe[''] = np.resize(cosFunctArray, temp_dataframe.shape[0])
        cosNormalizedZenith = temp_dataframe[''] * period_mean_tilt
        periodNormalizedZenith = cosNormalizedZenith.resample('M').mean()
        optimizedTiltAngles = apparentZenithArray.copy()

        for month in range(1, 13):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.month == month] =
periodNormalizedZenith[month - 1]

    if positonChangePeroiod == 'fortnightly' or positonChangePeroiod == 3:
        period_mean_tilt = apparentZenithArray.copy()
        period_mean_tilt.loc[period_mean_tilt > maxSurfaceTilt] = np.NaN
        temp_dataframe = pd.DataFrame(index=solar_position.index)
        cosFunctArray = pd.Series(
            [np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN,
np.NaN,
             1, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN, np.NaN,
             np.NaN])
        temp_dataframe[''] = np.resize(cosFunctArray, temp_dataframe.shape[0])
        cosNormalizedZenith = temp_dataframe[''] * period_mean_tilt
        periodNormalizedZenith = cosNormalizedZenith.resample('14D').mean()
        optimizedTiltAngles = apparentZenithArray.copy()

        for week in range(1, 55, 2):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.week == week] = periodNormalizedZenith[
                int((week - 1) / 2)]
        for week in range(2, 55, 2):
            optimizedTiltAngles.loc[optimizedTiltAngles.index.week == week] = periodNormalizedZenith[
                int((week - 2) / 2)]

    return optimizedTiltAngles, cosNormalizedZenith, periodNormalizedZenith


inputTiltAngles, cosNormalizedZenith, periodNormalizedZenith = calculate_mean_tilt('quarterly',
solar_position['apparent_zenith'], 75)

annual_energy = 0


def calculate_total_irrad(epw, solar_position, surface_tilt, surface_azimuth):
    dni_extra = pvlib.irradiance.get_extra_radiation(epw.index)
    total_irrad = irradiance.get_total_irradiance(
        surface_tilt=surface_tilt,
        surface_azimuth=surface_azimuth,
        dni=epw['dni'],
        ghi=epw['ghi'],
        dhi=epw['dhi'],
        dni_extra=dni_extra,
        solar_zenith=solar_position['apparent_zenith'],
        solar_azimuth=solar_position['azimuth'],
        model='haydavies')
    # model='isotropic')

    aoi = pvlib.irradiance.aoi(surface_tilt, surface_azimuth, solar_position['apparent_zenith'],
                               solar_position['azimuth'])
    airmass = pvlib.atmosphere.get_relative_airmass(solar_position['apparent_zenith'])
    pressure = pvlib.atmosphere.alt2pres(metadata['altitude'])
    am_abs = pvlib.atmosphere.get_absolute_airmass(airmass, pressure)
    temperature_model_parameters =
pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS['sapm']['open_rack_glass_glass']
    temps = pvlib.pvsystem.sapm_celltemp(total_irrad['poa_global'], epw['wind_speed'],
epw['temp_air'],
                                         **temperature_model_parameters)
    effective_irradiance = pvlib.pvsystem.sapm_effective_irradiance(
        total_irrad['poa_direct'], total_irrad['poa_diffuse'],
        am_abs, aoi, module)

    # v1
    # dc = pvlib.pvsystem.sapm(effective_irradiance, temps, module)
    # ac = pvlib.pvsystem.snlinverter(dc['v_mp'], dc['p_mp'], inverter)
    # ac = ac+ 2.328

    # v2 we consider a 4.5kWpeak installation (ca. 15 panels) to neutralize inverter losses
    dc = pvlib.pvsystem.sapm(effective_irradiance, temps, module) * 15
    ac = pvlib.pvsystem.snlinverter(dc['v_mp'], dc['p_mp'], inverter)
    ac = ac / 15
```

```
    # v3 losses are constant at 2.328 W/h
    # dc = pvlib.pvsystem.sapm(effective_irradiance, temps, module)
    # ac = pvlib.inverter.sandia(dc['v_mp'], dc['p_mp'], inverter)

    print('Generated energy:', str(int(ac.sum() / (module['Area'] * 1000))), 'kWh/m\u00b2')
    # print('Generated energy:',str(int(ac.sum())),'kWh/',str(module['Area']),'m\u00b2')
    annual_energy = ac.sum()

    return ac


df_monthly = pd.DataFrame()
df_tilts = pd.DataFrame()

# use apparent zenith as tilt
NS_tracking = solar_position['apparent_zenith'].copy()
# limit of 80deg
NS_tracking.loc[NS_tracking > 80] = 0

truetracking_angles = tracking.singleaxis(
    apparent_zenith=solar_position['apparent_zenith'],
    apparent_azimuth=solar_position['azimuth'],
    axis_tilt=0,
    axis_azimuth=180,
    max_angle=60,
    backtrack=False,  # False for true-tracking
    gcr=0.5)  # irrelevant for true-tracking

# get the module and inverter specifications from SAM

sandia_modules = pvlib.pvsystem.retrieve_sam('SandiaMod')
sapm_inverters = pvlib.pvsystem.retrieve_sam('cecinverter')
module = sandia_modules['Silevo_Triex_U300_Black__2014_']
inverter = sapm_inverters['SMA_America__SB7_7_1SP_US_40__240V_']

total_irrad_1 = calculate_total_irrad(epw, solar_position,
truetracking_angles['surface_tilt'].fillna(0),
                                      truetracking_angles['surface_azimuth'].fillna(0))

df_monthly['tracking'] = total_irrad_1.resample('m').sum()
df_tilts['apparent_zenith'] = solar_position['apparent_zenith']
df_tilts['azimuth'] = solar_position['azimuth']
df_tilts['elevation'] = solar_position['elevation']
df_tilts['tracking'] = NS_tracking
df_tilts['gIrr-tracking'] = total_irrad_1

for idx, word in enumerate(['quarterly', 'monthly', 'fortnightly']):
    temp_tilt = calculate_mean_tilt(word, solar_position['apparent_zenith'], 75)[0]
    total_irrad_irradiance = calculate_total_irrad(epw, solar_position, temp_tilt, 180)
    df_tilts[word] = temp_tilt
    column_name = f"gIrr-{word}"
    df_tilts[column_name] = total_irrad_irradiance[0]
    df_monthly[word] = total_irrad_irradiance.resample('m').sum()

fix = calculate_total_irrad(epw, solar_position, 40, 180)
df_monthly['fix'] = fix.resample('m').sum()
df_tilts['gIrr-fix'] = fix

annual_sum = df_monthly.resample('A').sum()
print('Annual Irradiance per selected tilt,azimuth:', str(int(annual_sum['fix'])))

df_monthly_kWh = df_monthly / 1000
df_monthly_kWh.plot.line()

# get current handles and labels
# this must be done AFTER plotting
current_handles, current_labels = plt.gca().get_legend_handles_labels()

# sort or reorder the labels and handles
new_labels = list(['E-W', 'S-1', 'M-1', 'M-2', 'F-35'])

# call plt.legend() with the new values
plt.legend(current_handles, new_labels)
plt.show()
# plt.savefig('/Users/dw/PycharmProjects/37project/Ontology/PV_panels/BS2021/images/santa_fe.pdf')

sns.lineplot(data=df_monthly_kWh, palette="tab10", linewidth=2)
plt.xlim(df_monthly.index.min(), df_monthly.index.max())
plt.xlim(df_monthly.index.min(), df_monthly.index.max())
plt.ylim(0, )
current_handles, current_labels = plt.gca().get_legend_handles_labels()

# sort or reorder the labels and handles
# reversed_handles = list(reversed(current_handles))
new_labels = list(['E-W', 'S-1', 'M-1', 'M-2', 'F-40'])
```

```
plt.legend(current_handles, new_labels)
plt.show()

df_monthly_kWh.index = pd.Series(['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
'Oct', 'Nov', 'Dec'])
sns.lineplot(data=df_monthly_kWh, palette="tab10", linewidth=2)
current_handles, current_labels = plt.gca().get_legend_handles_labels()

# sort or reorder the labels and handles
# reversed_handles = list(reversed(current_handles))
new_labels = list(['E-W', 'S-1', 'M-1', 'M-2', 'F-40'])

plt.legend(current_handles, new_labels)
plt.margins(x=0)
plt.grid(which='major', axis='x')
plt.ylabel("Monthly energy output [kWh/m\u00b2]")
# plt.savefig('/Users/dw/PycharmProjects/37project/Ontology/PV_panels/BS2021/images/AC_helsinki.pdf')
plt.show()
```