University of Zagreb

Faculty of Geodesy

Karlo Kević

An Ontology-based Model for Automated Spatiotemporal Disaggregation of Population Data

DOCTORAL DISSERTATION

Zagreb, 2025

University of Zagreb

Faculty of Geodesy

Karlo Kević

An Ontology-based Model for Automated Spatiotemporal Disaggregation of Population Data

DOCTORAL DISSERTATION

Mentor:
Asst. Prof. Ana Kuveždić Divjak, PhD

Zagreb, 2025

Geodetski fakultet

Karlo Kević

Ontološki model za automatiziranu prostorno-vremensku disagregaciju podataka o stanovništvu

DOKTORSKI RAD

Mentor:
doc. dr. sc. Ana Kuveždić Divjak

Zagreb, 2025.

GENERAL INFORMATION ABOUT THE DOCTORAL CANDIDATE

FULL NAME

Karlo Kević

DATE AND PLACE OF BIRTH

28 September 1994, Split, Croatia

CURRENT EMPLOYMENT

Research and Teaching Assistant at Faculty of Geodesy, University of Zagreb, Zagreb, Croatia

GENERAL INFORMATION ABOUT THE DOCTORAL DISSERTATION

TITLE

An Ontology-based Model for Automated Spatiotemporal Disaggregation of Population Data

SCIENTIFIC AREA / FIELD / BRANCH

Technical Sciences, Geodesy, Geomatics

MENTOR

Asst. Prof. Ana Kuveždić Divjak, PhD

NUMBER OF PAGES

210

NUMBER OF APPENDICES

4

NUMBER OF FIGURES

50

NUMBER OF TABLES

8

NUMBER OF REFERENCES

169

DOCTORAL THESIS EVALUATION AND DEFENCE

DATE OF THE PUBLIC DEFENCE OF THE DOCTORAL THESIS PROPOSAL:

19 December 2024

DATE OF THE DECISION ON THE APPROVAL OF THE DOCTORAL DISSERTATION PROPOSAL BY THE FACULTY COUNCIL OF THE UNIVERSITY OF ZAGREB FACULTY OF GEODESY:

30 January 2025

DATE OF THE DECISION ON THE APPROVAL OF THE DOCTORAL DISSERTATION PROPOSAL BY THE SENATE COUNCIL OF THE UNIVERSITY OF ZAGREB:

20 May 2025

APPOINTED THESIS EVALUATION COMMITTEE:

Assoc. Prof. Hrvoje Tomić, PhD*

Prof. Damir Medak, PhD*

Assoc. Prof. Martina Baučić, PhD**

* University of Zagreb Faculty of Geodesy

** University of Split Faculty of Civil Engineering, Architecture and Geodesy

DATA OF THE DECISION ON THE ACCEPTANCE OF THE POSITIVE REPORT OF THE THESIS EVALUATION COMMITTEE ON THE DOCTORAL THESIS BY THE FACULTY COUNCIL OF THE UNIVERSITY OF ZAGREB FACULTY OF GEODESY:

APPOINTED THESIS DEFENCE COMMITTEE:

DATE OF THE PUBLIC DEFENCE OF THE DOCTORAL THESIS:

STATEMENT OF ORIGINALITY

I declare that this doctoral thesis *An Ontology-based Model for Automated Spatiotemporal Disaggregation of Population Data* represents my original work and that I have used no other sources beside those that are referenced within it.

Zagreb, xx.xx.2025

Karlo Kević

Acknowledgments

Mentor

FULL NAME

Asst. Prof. Ana Kuveždić Divjak, PhD

POSITION

Chair of Geoinformatics, Institute of Cartography and Photogrammetry

AFFILIATION

Faculty of Geodesy University of Zagreb

ADDRESS

Kačićeva 26, 10 000 Zagreb

ORDID iD

0000-0003-1059-8395

CRORIS ID

28169

Abstract

High-resolution population data play a crucial role in modern decision-making processes. However, to comply with data privacy and statistical confidentiality requirements, population datasets are often spatially aggregated and released at relatively coarse spatial and/or temporal scales. This practice significantly limits data usability, prompting the scientific community to develop methods that enable temporal population estimation for areas smaller than the original reporting units. Existing disaggregation approaches rely on explicit methods based on spatial disaggregation techniques which often lack reusability and require extensive preprocessing and strong data manipulation skills. The research proposed in this thesis addresses these limitations by introducing the Population Disaggregation Ontology model (POPDO), model that semantically defines the disaggregation process using Semantic Web technologies. POPDO is an ontology built upon domain and task ontologies, designed to support automated, flexible and reusable spatiotemporal population disaggregation through semantic descriptions of the process workflow. The model adopts three-layer model architecture, POPDOd, POPDOr and POPDOp, which together facilitate a four-phase disaggregation process: temporal population data adjustment, weight computation, disaggregation and aggregation accross arbitrary spatial units. These layers encapsulate domain-relevant concepts and organize them into semantically coherent groups. While POPDOd and POPDOp define data domain concepts and process descriptions, POPDOr functions as an intermediary layer specifying the roles of domain data within the disaggregation method. Testing the ontology on real world case study proves POPDO ontology to be applicable for population disaggregation and highlights its potential as a viable alternative to traditional disaggregation approaches. In an era of increasing availability of semantically described geospatial and statistical data, the POPDO ontology provides a missing piece that enables more effective and efficient use of population disaggregation methods.

Sažetak

Podaci o stanovništvu visoke prostorne razlučivosti imaju ključnu ulogu u suvremenim procesima donošenja odluka. Kako bi se ispunili zahtjevi zaštite osobnih podataka i statističke povjerljivosti, skupovi podataka o stanovništvu često se prostorno agregiraju i objavljuju na relativno grubim prostornim i/ili vremenskim razinama. Takva praksa značajno ograničava iskoristivost podataka, što potiče znanstvenu zajednicu na razvoj metoda koje omogućuju vremensku procjenu prostorne razdiobe stanovništva za područja manja od izvornih referentnih jedinica. Postojeći pristupi disagregaciji uglavnom se oslanjaju na eksplicitne metode temeljene na tehnikama prostorne disagregacije, koje često imaju ograničenu mogućnost ponovne upotrebe te zahtijevaju napredne vještine obrade podataka. Istraživanje predstavljeno u ovoj disertaciji doprinosi navedenim ograničenjima uvođenjem ontološkog modela za prostorno vremensku disagregaciju podataka o stanovništvu (POPDO), koji semantički opisuje proces disagregacije koristeći tehnologije Semantičkog weba. POPDO je ontologija izgrađena na domenskim i procesnim ontologijama, osmišljena da podrži automatiziranu, fleksibilnu i ponovno upotrebljivu prostorno-vremensku disagregaciju podataka o stanovništvu putem semantičkih opisa izvođenja procesa. Model se temelji na troslojnoj arhitekturi, POPDOd, POPDOr i POPDOp slojevima koji zajednički omogućuje provedbu procesa disagregacije kroz četiri faze: vremensko usklađivanje podataka o stanovništvu, određivanje prostornih težina, disagregaciju te agregaciju na proizvoljne prostorne jedinice. Predloženi slojevi obuhvaćaju koncepte relevantne za domenu disagregacije i organiziraju ih u semantički koherentne skupine. Dok POPDOd i POPDOp opisuju koncepte domene podataka i procesa, POPDOr djeluje kao međusloj koji domenskim podacima dodjeljuje uloge unutar metode disagregacije. Rezultati testiranja ontologije na primjeru disagregacijske metode pokazali su da je POPDO model primjenjiv za disagregaciju populacije te su istaknuli njegov potencijal kao održivu alternativu tradicionalnim pristupima disagregaciji. U razdoblju sve veće dostupnosti semantički opisanih geoprostornih i statističkih podataka, POPDO ontologija predstavlja nužnu sponu koja će omogućiti učinkovitiju i djelotvorniju primjenu metoda disagregacije na podatke o stanovništvu.

CONTENT

List of Figures

List of Tables

Introduction

Addressing complex human–environment interactions, such as the allocation of food and medical supplies, transportation planning, risk prevention, and the management of natural hazards, requires detailed, time-specific information on the geographical distribution of populations (Galvani et al., 2016; Calka et al., 2017). However, in order to meet data privacy and statistical confidentiality requirements, population datasets are typically spatially aggregated and disseminated at relatively coarse spatial and/or temporal resolutions (Batsaris & Zafeirelli, 2023). Consequently, high-resolution population data are often unavailable and are frequently derived through spatial and temporal disaggregation of existing datasets.

The disaggregation of population data is experiencing constant changes, as researchers seek to develop improved models that more accurately represent the spatial and temporal distribution of population. The advent of the digital technologies has significantly transformed the methodologies employed in this domain, leading to substantial changes in disaggregation practices. For instance, computational processes are now executed within digital environments, enabling faster and more efficient performance. Modelling techniques have expanded, allowing for the implementation of more sophisticated and comprehensive disaggregation strategies. Moreover, the acquisition and dissemination of a wide range of spatially referenced datasets capable of indicating potential concentrations of population has become increasingly feasible. Collectively, these developments underscore the critical role of technology in the disaggregation process and prompt further consideration of how its full potential might be most effectively realised.

The growing demand for user-defined resolution of population data, coupled with the increasing availability of spatial datasets suitable for disaggregation, highlights the need for an automated and flexible solution capable of supporting a range of disaggregation approaches. To meet these requirements, such a solution must be adaptable rather than static, not bound to a specific technological platform, and able to overcome the limitations posed by heterogeneous data sources, e.g. such as the integration of varying data structures. Furthermore, it should be capable of modelling the disaggregation process as a structured sequence of steps. This would enable broader applicability, enhance the replicability of results, reduce the need for manual data preprocessing, and allow full

automation of the process from start to finish. One promising approach involves the development of a conceptual model grounded in ontology, drawing upon the advantages of semantic data integration and the extraction of implicit knowledge within the domain of spatial disaggregation. A model of this kind appears well-suited to fulfilling the outlined requirements, and its development therefore constitutes the principal focus of this research.

1.1 Relevance and Limitations of Population Data

The wide applicability of population data stems from the complex and multilayered nature of human–environment interactions, where detailed demographic information supports more informed and effective decision-making. For instance, planning of sustainable and compact urban areas requires local amenities to be in a walking-distance from residents, so availability of e.g. block/district population data is needed to assess suitability of urban spaces and propose optimal locations for new amenities (Fina et al., 2022; Jama et al., 2025; Telega et al., 2021). Similarly, effective risk management in both urban and rural contexts, such as the mitigation of landslides, rockfalls, or air pollution, relies heavily on detailed population data. In the event of a hazard, which can occur on a specific section of an urban area, population data at the level of an entire statistical unit is often inadequate. Targeted prevention, impact assessment and response planning require knowledge of a number of people living in affected zone (Li, 2022). The importance of high-resolution population data extends beyond urban planning, with significant relevance to diverse domains. In scientific research, e.g. in observational health studies and epidemiology, more accurate spatial population data enables precise calculation of disease rates and risk assessments, thereby reducing biases introduced by outdated or overly aggregated datasets (Fecht et al., 2020; Wang & Wang, 2024). In commercial sectors, such as marketing and retail, fine-scale demographic information enhances the effectiveness of recommendation systems by enabling personalised promotions and the tailoring of products or services to the actual location of target populations. This spatially informed approach contributes to more context-aware marketing strategies and can ultimately improve sales performance (Shili & Sohaib, 2025).

In the European Union, the relevance of population data is formally recognized in legislation, and it is subject to special regulatory treatment. According to EU Implementing Regulation 2023/138, official population data, classified as statistical data, is designated as high-value data, reflecting its capacity to generate anticipated social and economic benefits when made accessible for reuse (European Commission, 2023). Consequently, the regulation stipulates that such data must be disseminated at the highest possible level of granularity, while simultaneously ensuring compliance with legal safeguards, including the protection of personal data This classification underscores the critical importance of population data in supporting evidence-based policymaking and strategic planning, which is why the European Commission mandates it to be provided as open data, freely accessible and free from restrictive licensing (European Parliament, 2019).

While this legal framework has undoubtedly affirmed the importance of official granular population data and enhanced its availability across Europe, its practical usability remains limited. Constraints such as data privacy regulations, political considerations, and underdeveloped data infrastructures often result in population data being aggregated to statistical or administrative spatial units. This level of aggregation impedes its applicability, as decision-making processes frequently concern smaller geographic areas rather than entire administrative units for which data is typically provided (Stevens et al., 2015). Moreover, changes in administrative boundaries over time compromise the comparability of statistical data, posing a significant barrier to longitudinal analyses in which population data plays a crucial role (Bernard et al., 2022).

Bridging the gap between the limitations of existing population data and the requirements of practical applications necessitates a solution capable of delivering user-defined levels of granularity. Such a solution would enable more responsive and better-informed decision-making by generating data that more accurately reflects the dynamic interactions between populations and their environments.

1.2 Approaches to Process Modelling

Spatiotemporal disaggregation methods may be conceptualised as a sequence of steps, a process involving the integration of diverse input datasets to generate a specific output.

Two potential approaches can be considered for modelling such a process: Geographic Information Systems (GIS) and ontology-based modelling. Given that the primary requirement for a modelling approach is to enable the automated execution of varied disaggregation workflows involving heterogeneous spatial data, the chosen framework must be sufficiently flexible to accommodate multiple input sources and remain method-agnostic in order to support a broad spectrum of disaggregation techniques.

GIS systems are the most widely used approach for geospatial data integration, as they are specifically designed to perform complex spatial analyses (Oliveira et al., 2024; Wieczorek & Delmerico, 2010). However, GIS data model is syntactic and schema-based, which introduces notable constraints when dynamic or adaptive applications are required. While GIS tools can be employed for population disaggregation, several limitations hinder their broader applicability. Firstly, GIS workflows are typically constructed manually or rely on predefined scripts. Although this offers a degree of flexibility, such workflows are inherently task-specific and static, and any methodological modification requires reconfiguration or rewriting of the script. Secondly, employed scripts are often tightly coupled with specific data structures, which significantly restricts their reusability when datasets vary in format. Thirdly, while GIS excels in managing spatial data, the integration of non-spatial datasets is frequently cumbersome, often requiring ad hoc solutions that undermine the practicality of seamless data integration. Finally, GIS systems do not natively support formal logic or reasoning capabilities, which limits their ability to validate processing steps or infer implicit relationships, such as enforcing value preserving constraints. As a result, the establishment of contextual relationships between datasets and the interpretation of metadata generally rely on manual intervention by the human operator.

The ontology-based approach provides a straightforward yet robust data modelling framework that facilitates the integration and representation of heterogeneous datasets and their interrelationships (Sun et al., 2019). By focusing on data semantics rather than data structure, it offers a means of overcoming heterogeneousness across diverse data sources. Unlike GIS, which conducts spatial analysis through the quantitative representation of real-world phenomena using geographic coordinates, ontologies employ

reasoning over qualitative descriptions to draw inferences (King, 2019). This allows spatial relationships to be captured semantically within the model, enabling spatial analysis to be executed through ontology-supporting tools. The simplicity and adaptability of the ontology data model allow for the seamless integration of new datasets without requiring modification of the existing structure. Furthermore, qualitative reasoning enables a broad range of inferencing possibilities, which may be aligned with different methodological requirements. Semantic definitions embedded in the model also support the extraction of implicit knowledge directly from the data, reducing reliance on user interpretation. While ontology reasoning relies on query languages that do not inherently support control flow or scripting, such procedural logic can be provided by linking graph database with external technologies. Taken together, these advantages position the ontology-based approach as a compelling alternative to GIS for modelling diverse data inputs across a variety of population disaggregation methodologies.

1.3 Objectives and Hypotheses

Population disaggregation is heavily grounded in the spatial component of the data which is why the focus of scientific community in the domain is pointed at developing methods that better describe spatial correlation of population-indicating data and can produce more accurate population distribution. While such research undoubtedly enhances theoretical foundations of population disaggregation, Flasse et al., (2021) argue that practical tools for disaggregation are needed to ease the access to more detailed population data. Furthermore, the increasing demand for timely user-defined granulation of population data coupled with the increasing availability of relevant datasets for disaggregation requires automated solution that minimises human intervention and offers human independent processing. De Meester et al. (2020) associate this with the benefits of semantic web. Semantic web applications are becoming vital tools for decision making, with semantic descriptions which are crucial for their automated execution (de Souza Neto et al., 2018; Bednar et al., 2024). These considerations underscore the need to identify a viable solution, which forms the principal goal of this research:

> *"to develop an ontological model for flexible and automated spatiotemporal disaggregation of population data onto arbitrary spatial tessellations using semantic web technologies."*

Achieving this goal is closely tied to four specific objectives that will ensure sustainability of the proposed solution:

> *"Define key concepts and spatial relations in spatial disaggregation methods."*

Spatial disaggregation methods vary in their approaches to generating disaggregated data; however, they all share a common underlying principle. By generalising the process and focusing on the key components and spatial relationships, the proposed model will possess sufficient flexibility to accommodate a range of disaggregation techniques.

> *"Ensure interoperability of the proposed ontological model."*

Semantic data modelling is well established within the data domain, with numerous widely adopted standards for describing various types of data. Additionally, process modelling through ontologies has recently gained traction, with several ontology models now available. The research community strongly advocates for the development of models that build upon existing knowledge rather than creating isolated, standalone solutions, as this approach ensures broader applicability and enhances the long-term sustainability of the model.

> *"Develop ontology model for modelling spatial disaggregation procedures."*

Population disaggregation is a methodological process comprising several sequential steps required to achieve the desired outcome. Although ontology models typically offer a static representation of the domain through key concepts and relationships, the intention here is to utilise these models to describe the procedural workflow within a static framework. In doing so, the model will remain consistent with the principles of ontology while effectively supporting process of disaggregation.

"Test the developed disaggregation model."

The development of an ontology model is an iterative process that progressively uncovers modelling limitations at each stage. Addressing these limitations through successive testing phases ensures that the model is suitably robust to support flexible and automated spatiotemporal disaggregation.

The proposed ontology model aims to address the challenge of providing population data at user-defined spatiotemporal granularities. The development of this model is anticipated to make significant contributions to both theoretical and practical domains. Theoretically, the research will extend the existing knowledge base by offering semantic descriptions of the procedures involved in spatiotemporal disaggregation of population data, which are fundamental for the advancement of AI-driven automated solutions. Practically, the research is expected to facilitate the automation of population data disaggregation across arbitrary spatial tessellations through the utilisation of semantics embedded within the ontological framework.

The research objective is founded on the premise that semantic web technologies provide the necessary tools to facilitate automated population disaggregation. To test this premise, two hypotheses have been formulated:

1. *Methods of spatial disaggregation for population data can be conceptually modelled as procedures in an ontological model, where procedures, spatial relations, and key components, such as input data, parameters, and outputs — are defined.*

2. *Modelling spatial disaggregation methods as ontological procedures, it is possible to automate the disaggregation of population data.*

1.4 Out of Scope

The main goal of this research, along with its specific objectives, centres on establishing a proof of concept for an ontology model designed to support spatiotemporal population disaggregation. Consequently, the focus is not on developing novel disaggregation methodology to enhance accuracy, but rather on creating a broadly applicable semantic

model capable of facilitating automated spatiotemporal population disaggregation. Accordingly, the model does not demonstrate the disaggregation of population data itself but instead illustrates how the disaggregation process can be represented within a semantic web environment.

The proposed ontology model is method-agnostic and intended to support a range of population disaggregation approaches. Given that disaggregation methods differ in both the number and type of datasets required, this research aims to deliver a universal solution capable of accommodating a broad spectrum of methodologies and input data types. Accordingly, the focus is placed on the development of a generic, high-level model that is widely applicable and not restricted to any single disaggregation technique. However, to demonstrate proof of concept, the model will be tailored to the requirements of a simple methodological approach and evaluated within that context.

The proposed model must be capable of incorporating a temporal component to enable the estimation of population figures beyond the reference timestamp. However, to ensure broad applicability, time will be treated as an attribute of spatial data rather than an integral element of the data structure. This approach will allow the disaggregation model to produce meaningful results without necessitating high temporal resolution in the input data. In doing so, the model will remain universally applicable and functional even in contexts where detailed spatiotemporal datasets are not readily available. On the other side, the model will not consider changes of spatial units in time and will consider them static for the referent timestamp of population data.

The model will be tested using publicly available data in its original, unaltered form. Rather than integrating the data into the model itself, each dataset will be treated as an independent source, accessed dynamically through queries as needed. This approach preserves the original data structure, minimises preprocessing requirements, and facilitates the reuse of existing semantically described datasets. Furthermore, for the sake of simplicity, the model will treat all data as static and will not include mechanisms for harvesting dynamic data from data providers.

1.5 Thesis Structure

This thesis is structured in five chapters, as outlined below:

Chapter 2 provides the research background, introducing the key concepts within the domain. It examines the geospatial characteristics of population data that make the application of spatial disaggregation methods possible. The chapter also includes a comprehensive literature review of existing population disaggregation techniques, highlights their potential for procedural modelling.

Chapter 3 focuses on Semantic Web technologies, particularly the use of ontologies. It contrasts the Semantic Web with the traditional web of documents, outlining the advantages of the former. The chapter also introduces the principal technologies of the Semantic Web stack and offers an in-depth explanation of the concept of ontologies, including a taxonomy of ontology types and a review of existing domain ontologies relevant to this research. This chapter provides state of the art in population disaggregation based on ontology approach and gives an overview of ontologies for specific parts of the disaggregation process.

Chapter 4 details the steps involved in ontology development. It begins with an analysis of user requirements that guide the ontology design. This is followed by the development process itself, where the different levels of modelling are described.

Chapter 5 presents the testing and evaluation of the proposed methodology. It describes the data sources and preprocessing steps, and it illustrates the execution of the disaggregation process on a concrete disaggregation method. This chapter delivers results and enables the validation of various components of the model.

Chapter 6 offers a discussion on the proposed model, examining its benefits and limitations. It also assesses whether the developed model has contributed to the achievement of the research objective based on the hypotheses set out.

Finally, Chapter 7 outlines main conclusions of the proposed disaggregation ontology and provides directions for future research.

Spatiotemporal Disaggregation of Population Data

This chapter aligns proposed research with the foundational spatiotemporal disaggregation concepts by providing review of relevant scientific achievements in field of population disaggregation. It begins with an overview of the theoretical concepts related to population data and the digital representation of geospatial information, introducing the topic and its foundations. Following, it examines common methods for spatial and temporal disaggregation, as well as existing practical implementations. Together, these topics establish the background for the conceptual model of spatiotemporal population disaggregation developed in this research.

2.1 Concept Overview

2.1.1 Geospatial Data in Digital Environment

When perceiving and describing the real-world, people tend to parse it into a set of discreet entities rather than describing it as a continuous space. This tendency may have evolutionary roots as describing location and properties of specific objects, such as food or predators, likely had greater survival value back in history (Peuquet et al., 1998). It may also explain why natural language is generally better suited to describing discreet objects than continuous fields in geography (Cova & Goodchild, 2002).

In GIS, the two primary geographical data models for representing real-world phenomena are the object and field models. These are not inherent properties of the phenomena themselves, but rather conceptual perspectives adopted by the user for a specific application (Cova, 2017). While most geographic entities are typically perceived as either objects or fields, this dichotomy is not absolute. For example, a person may be treated as an individual object, but when considered as part of a population, the discrete object characteristic disappears, and the person becomes a value within a population density field (Cova & Goodchild, 2002). This illustrates that any geographic reality can be conceptualized using either model. However, because natural language lacks the precision required for computational environments, the distinction between the concepts of object and field must be clearly defined before using geospatial data in a computer-based context (King, 2019).

According to the field conceptualization, a geographic phenomenon is represented as a continuous surface whose values vary with location (Jacquez et al., 2000). Every point

within an infinitely dense space, a field, is associated with a value drawn from an attribute domain (e.g., temperature, elevation, moisture), making the representation inherently functional (Cova, 2017). For this reason, a field can be described as a single-valued function of space (Cova & Goodchild, 2002). Fields are particularly well suited for representing phenomena without clearly defined boundaries, instead exhibiting smooth or heterogeneous variation across space. This conceptualization brings several defining characteristics: values are theoretically available for every location, fields are spatially continuous and conceptually infinite, and the spatial frame of reference can be one-, two-, three-, or four-dimensional (space and time). Attributes are tied to a specific domain (e.g., precipitation) and can be expressed using various measurement scales, including binary, nominal, ordinal, interval, and ratio (Cova & Goodchild, 2002). In practice, however, field representation in a digital environment must be approximate, as it is impossible to store values for every location (Goodchild, 1992). To address this, spatial tessellations, regular or irregular point networks, and contour representations are used to approximate continuous reality within finite storage (Cova & Goodchild, 2002). Missing values are then calculated from the stored approximation using spatial interpolation techniques.

In the object conceptualization, the real world is viewed as composed of discrete, spatially distinct objects with precise and unbiased locations and extents (Kjenstad, 2006), like lakes and roads, which are distinct in their identity and attributes (Cova & Goodchild, 2002). Smith & Mark (1998) categorize these objects based on their boundaries into three types: physical objects with clear cognitive boundaries, such as rivers and bridges; geographic objects, like bays and mountains, whose extents are partially tied to the physical world but also shaped by human delineation; and geopolitical objects, such as statistical and administrative units, which are purely product of human cognition. Based on boundary types, Smith & Mark (1998) further classify objects as *fiat* (human-defined) or *bona fide* (naturally defined). Regardless of boundary type, the clear demarcation of objects from their surroundings makes them well-suited for representation in GIS using common geometric forms such as points, lines, or polygons.

Due to factors such as personal data protection concerns or the focus on broader population trends rather than individuals, the field conceptualization is most commonly used in population studies. Typical outputs include choropleth maps or population density maps, where population values are aggregated and expressed for spatial units like enumeration areas, or grid cells. Because numerous spatial tessellations are used to create density and choropleth maps, these are considered discrete value *fiat* fields, where the values depend on the type and size of the spatial units employed.

2.1.2 Population Data

Although intuitively understood, the meaning of population data can sometimes be ambiguous, especially when compared to related terms such as demographic data. Demographic data refers to specific characteristics within a population, such as age, sex, or race (Nettleton, 2014), whereas population data encompasses broader statistical information about a group of people (Christen & Schnell, 2023), typically quantitative measures of the number of individuals within a given area (United Nations Department of Economic and Social Affairs, 2017).

Population data is most often collected at the resolution of individual persons for administrative or operational purposes (Christen & Schnell, 2023). This means that a variety of characteristics are gathered and stored for each individual in the dataset. However, the dissemination of such highly detailed data is limited by several factors. These include underdeveloped data infrastructures that cannot support the distribution of detailed datasets (Comber & Zeng, 2022) as well as political manipulation of population information for specific agendas (Espey et al., 2025). More commonly, concerns over privacy and the risk of exposing sensitive personal information are cited as the primary barriers, since detailed data could jeopardize individuals' privacy and facilitate malicious use (Lin & Xiao, 2024; Lloyd et al., 2019; United Nations Department of Economic and Social Affairs, 2022). This barrier makes it challenging for data providers to meet the needs of public good while maintaining privacy of individual data which requires mechanisms to balance between the two. Aggregation of population data is a common approach to bridge this gap by providing data at a coarser level of detail than originally collected.

Although population data is primarily statistical, its spatial component plays a significant role in shaping its interpretation and use. For instance, data contextualization relies heavily on the spatial extent of the data (Matthews & Parker, 2013; Raymer et al., 2019) or, in simple words, stating the number of people without spatial reference offers little value to the end user. Additionally, geographically aggregated data is directly influenced by the size and shape of the spatial units involved (Bernard et al., 2022; Briant et al., 2010; Fendrich et al., 2022) and different spatial tessellations can lead to variations in the resulting aggregated data. This property is leveraged in gridded population products, where the arbitrary resolution of the grid structure helps preserve individual anonymity, supports higher spatial resolution in aggregated data, and mitigates issues related to modifiable spatial boundaries that hinder data comparability over time (Leyk et al., 2019; Monteiro et al., 2021; United Nations Department of Economic and Social Affairs, 2021). On the other hand, the spatial component facilitates the integration of population data with other spatially referenced datasets and fosters the development of *spatial demography*, a statistical field that employs spatial techniques to better understand demographic processes across space (Raymer et al., 2019).

Population data typically originates from official sources such as household sample surveys, administrative registers, and most commonly, population and housing censuses (Suharto, 2001; United Nations Department of Economic and Social Affairs, 2008) and is aggregated to administrative or statistical units to meet administration requirements (Christen & Schnell, 2023). While this is good enough for social applications, from spatial perspective such aggregation is misleading. Administrative or statistical boundaries often do not align with the physical discontinuities of the variables of interest, creating an illusion of uniform spatial distribution that is commonly visualized through quantitative choropleth maps (Lloyd et al., 2019). Nevertheless, such aggregated data serves as the foundation for many globally available datasets, such as WorldPop, which aim to produce more accurate and detailed representations of population distribution.

2.2 Spatial Population Disaggregation

As a result of aggregation, population data is tied to fixed spatial boundaries. To address practical needs, it is often necessary to redistribute the population data more accurately across space.

2.2.1 Population Disaggregation

The spatial component of population data enables the use of spatial techniques for redistributing population counts. This process is known as spatial disaggregation and refers to reallocating spatially aggregated population data from coarse administrative units, source zones, into finer spatial units, target zoness, to better reflect spatial heterogeneity of human distribution (Figure 2.1). Spatial disaggregation relies on areal interpolation, a specialized form of spatial interpolation used to transfer data between different sets of spatial boundaries (Lam, 1983), commonly applied for data comparison or integration (Eicher & Brewer, 2001). Even though formally based on interpolation techniques, application role of areal interpolation and the fact that data values represent count, and not density, makes population redistribution considered as disaggregation rather than interpolation (King, 2019).

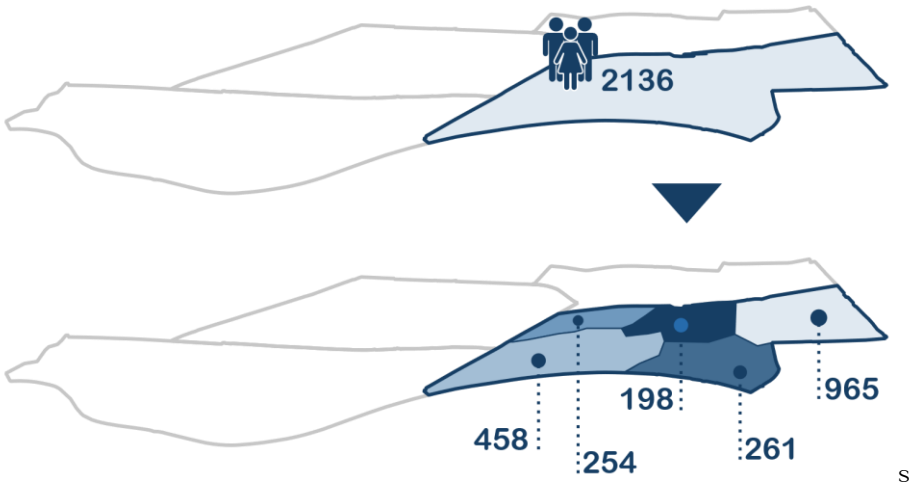


Figure 2.1
Illustration of general principle of spatial disaggregation of population data from reference spatial unit to lower-level spatial units

Areal interpolation methods used in the spatial disaggregation of population data depend on a weighting layer that guides how the population is distributed across target zones

(Cheng et al., 2022; Stevens et al., 2015). Over time, new approaches to disaggregation created refined methods that resulted in more sensitive allocation procedures. The literature classifies these methods in various ways, often based on the types of inputs used to guide disaggregation. For instance, Qiu et al. (2022) distinguish between areal interpolation and dasymetric mapping, while Leyk et al. (2019) identify four categories: areal weighting techniques, dasymetric mapping, statistical methods, and hybrid approaches. Netrdová et al. (2020) divide methods into cartographic and geostatistical, Wu et al. (2005), as cited in Calka et al. (2016), separate them into areal interpolation and statistical modelling, and Comber & Zeng (2019) differentiate between methods that use ancillary data and those relying solely on source and target zone properties. Because many methods fit within the framework proposed by Comber & Zeng (2019), this classification is widely adopted (e.g. Sapena et al., 2022) and forms the basis for further discussion in this work.

2.2.2 Areal Disaggregation Using Zone Spatial Characteristics

This group of disaggregation methods is zone oriented, meaning they use characteristics of the spatial zones themselves to guide the redistribution process. These methods are rather simple, often used in lack of other informative geodata and based on the assumption of homogeneous distribution of population in the source zone (Sapena et al., 2022). The two fundamental approaches in this category are simple areal weighting and pycnophylactic interpolation.

Simple areal weighting is the most straightforward disaggregation method. It redistributes population data based on the proportion of the overlapping area between each source zone and target zones (Sapena et al., 2022) (Figure 2.2). Because weights are derived from area shares, the method is inherently volume-preserving, meaning the sum of the disaggregated data across all target zones equals the original input count. The method needs no ancillary data and requires accurate boundary geometries only which makes it easily implemented in computer environment using polygon overlay operations (Comber & Zeng, 2019). As concluded by Goplerud (2016), in certain situations, such as transferring election results across changing statistical units, this method can provide good accuracy, with a mean absolute error of 2–3%. However, a key limitation is its assumption

of spatial homogeneity within target zones, which rarely holds true in practice (Monteiro et al., 2021). Nonetheless, when additional geospatial data is unavailable, simple areal weighting remains a practical and reasonable solution (Comber & Zeng, 2019).

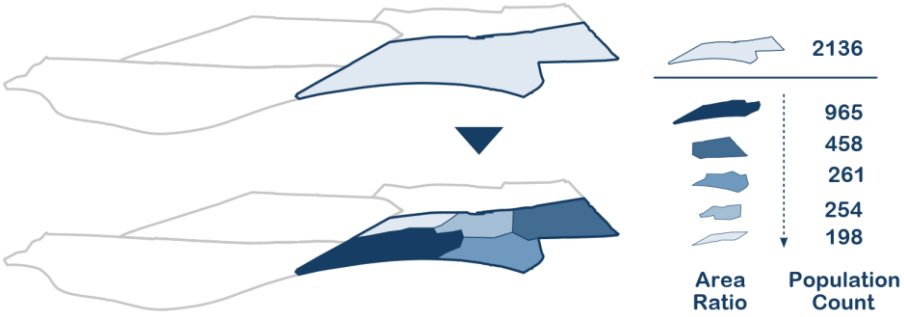


Figure 2.2
Illustration of areal weighting disaggregation method approach

Pycnophylactic interpolation differs from simple areal weighting but is considered a refinement of this volume-preserving approach (Qiu et al., 2022). The idea is to create smooth surface in the target zones, often raster cells, from counts in source zones, typically polygons (Tobler, 1979). It begins by applying volume-preserving areal weighting (like in previous method) to obtain initial values for the target zones. These values are then smoothed by replacing each with a weighted average of its nearest neighbours. After smoothing, the total values of the target zones are compared to the original source zone counts and adjusted to maintain volume preservation (Figure 2.3). The process is repeated until smooth surface with no sharp discontinuity in adjacent target zones is achieved. Number of iterations is not fixed, and the process usually ends when difference in values between two consecutive iterations is small enough (Monteiro et al., 2021). According to Comber & Zeng (2019) pycnophylactic interpolation is well-suited for creating surfaces from count data, but assumption of continuity, i.e. no sharp boundaries in the distribution may not always hold true, especially in cases where physical barriers like waterways or roads segment the area.

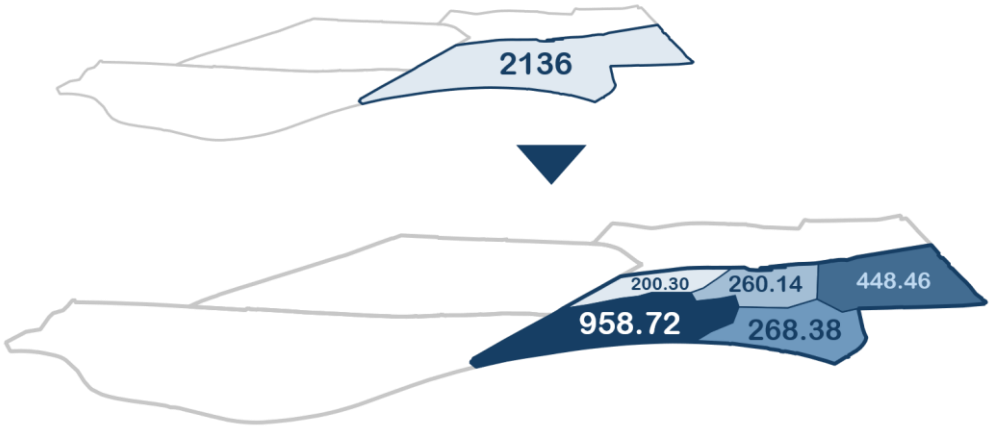


Figure 2.3
Illustration of pycnophylactic disaggregation method approach

Inconsistencies such as redistributing population into uninhabited areas, like bodies of water or forests, often arise when using earlier mentioned disaggregation methods, limiting their usefulness for detailed decision-making at sub-census levels (Stevens et al., 2015). On the other hand, the increasing availability of Earth observation data, including nighttime lights, land cover, digital terrain models, and other geospatial datasets like addresses or land use classes, provides a wealth of variables that can help identify likely population locations and thus constrain the allocation process (Calka et al., 2016; Liu et al., 2023; Sapena et al., 2022; Ural et al., 2011; Weber et al., 2018). However, as Stevens et al. (2015) point out, effectively using this data to produce more accurate population distributions requires the development of more advanced disaggregation methods, which fall under the second category, methods that incorporate ancillary data.

2.2.3 Areal Interpolation Using Ancillary Data

The diversity of available ancillary data and the ways it is combined in the disaggregation process have led to a wide variety of disaggregation methods. Some authors (e.g. Cartagena-Colón et al., 2022; King, 2019; Mennis & Torrin, 2005; Netrdová et al., 2020; Qiu et al., 2022) associate such methods exclusively with dasymetric mapping and distinguish between its various subtypes. In contrast, other authors, such as Stevens et al. (2015), Sapena et al. (2022), Leyk et al. (2019) and Comber & Zeng (2019) consider dasymetric mapping as just one of methods within this broad category.

Differences in how disaggregation methods are grouped often stem from varying interpretations of the term *dasymetric mapping*. As introduced by the Russian cartographer Tian-Shansky, dasymetric mapping aims to overcome the limitations of the homogeneity assumption in choropleth population maps by subdividing areas into smaller partitions where this assumption is more valid, typically using external data (Bielecka, 2005). Different interpretations mainly arise from how ancillary data is applied and the nature of the resulting spatial units. For some authors, e.g. Qiu et al. (2022), any operation with ancillary data, mathematical or statistical, that can characterize its relationship with population and result in small areas with similar distribution patterns is considered dasymetric mapping. Conversely, Comber & Zeng (2019), and Sapena et al. (2022) consider a narrower definition, restricting dasymetric mapping to cases where ancillary data serves as spatial control to include or exclude areas from redistribution, while more complex methods, those that do not necessarily produce uniform population densities, are categorized as statistical or other approaches. These different views may be based on cartographic background of the term *dasymetric* which usually refers to general type of thematic map created using different methods (Eicher & Brewer, 2001).

The literature suggests that practical disaggregation approaches often combine multiple methods to improve the accuracy of results (Cartagena-Colón et al., 2022; Hallot et al., 2021; Stevens et al., 2015). To clearly differentiate between these approaches, methods using ancillary data will be explained following the classification proposed by Sapena et al. (2022); dasymetric mapping, statistical methods and hybrid methods.

Dasymetric Mapping

Mennis & Hultgren (2006a) describe dasymetric mapping as a method similar to areal interpolation as both involve transforming data from arbitrary source zones to different geometrically defined target zones. However, the key distinction lies in the use of ancillary data in dasymetric mapping, which helps characterize the relationship between population data and the target regions (Mennis & Hultgren, 2005), as well as the omission of a subsequent data reaggregation step to preferred spatial units in dasymetric mapping (Eicher & Brewer, 2001). According to Mennis (2003) dasymetric mapping combines areal weighting (based on intersection areas) with relative densities derived from ancillary

spatial classes to spatially redistribute population data. Šveda et al. (2024) state that this method yields high-quality results, as demonstrated by its successful applications in diverse fields such as tourism, accessibility assessment, and crisis management. In this approach, population redistribution is guided by weights derived from ancillary data, and based on how this data is utilized, the literature identifies three main types of dasymetric mapping.

Binary Dasymetric Mapping

Binary dasymetric mapping functions as a weighting mask that uses ancillary data to differentiate between populated and unpopulated areas (Comber & Zeng, 2022) (Figure 2.4). Unpopulated regions are assigned a weight of zero and thus excluded from the disaggregation process, while the entire population is allocated to areas deemed populated with a weight of one (Su et al., 2010). A similar distinction can be made between urban and rural zones, where population density differences are obvious (Cromley et al., 2011). This method most commonly utilizes land use classes to identify urban spaces, though in some cases, expert knowledge or manually defined rules are applied (Monteiro et al., 2021), which may affect the reliability of results. The simplicity of the binary mask approach facilitates easy implementation, and Qiu et al. (2022) note that it effectively addresses the common issue of underestimating population in urban areas and overestimating it in rural areas.

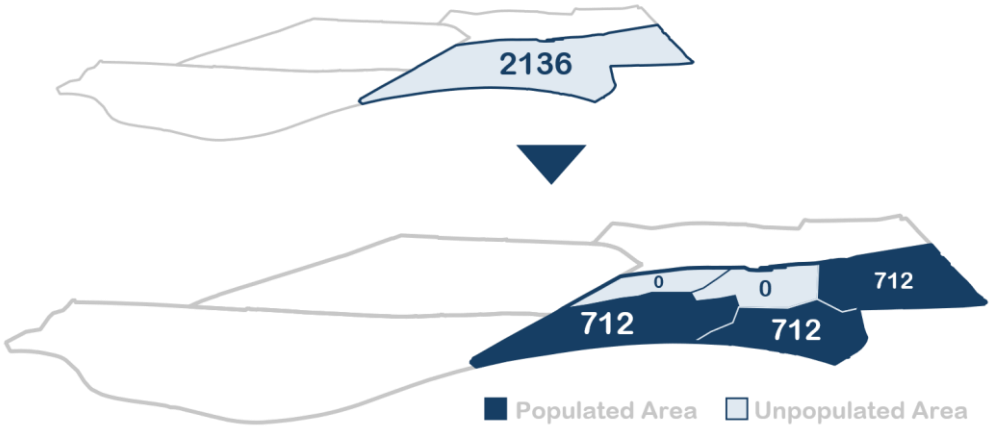


Figure 2.4
Illustration of binary dasymetric disaggregation method approach

Multi-class dasymetric mapping

The simplicity of the previous method stems from its binary division of the source zone, which often does not reflect real-world complexity. Multi-class dasymetric mapping improves on this by introducing multiple subzones, each assigned a different weight (Su et al., 2010) (Figure 2.5). These weighted gradations, rather than just 0 and 1, tend to yield more accurate population redistribution by incorporating the assumption that more urbanized areas have higher population densities, and less urbanized areas have lower densities (Qiu et al., 2022). Subzones are typically defined by the researcher based on local knowledge or extracted from land cover data (Qiu et al., 2022; Sapena et al., 2022). However, the weights assigned to each subzone are provided by the operator and often expressed as percentages that preserve the total population volume (Su et al., 2010). Ultimately, the total population is allocated across the subzones according to assigned weights.

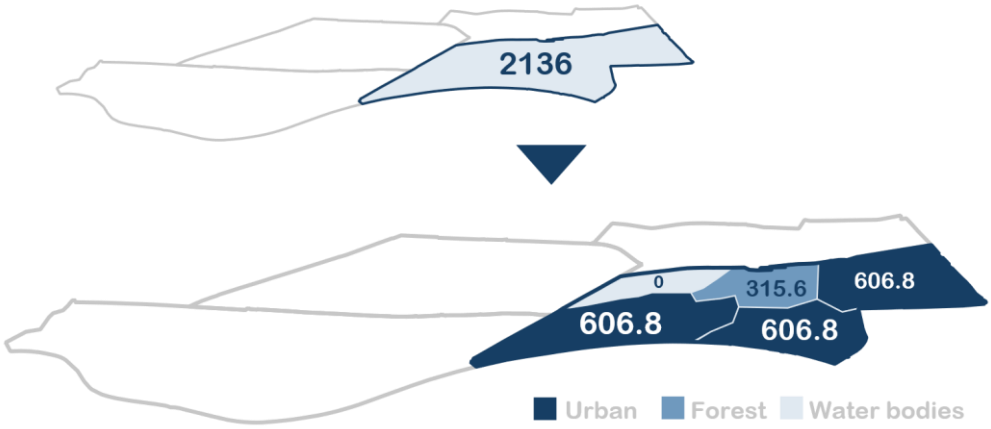


Figure 2.5
Illustration of multi-class dasymetric mapping method approach

Multi-class dasymetric mapping can provide higher quality results than the binary approach; however, its major limitation lies in the subjective assignment of weights to subzones. Additionally, Su et al. (2010) point out that the method assumes uniform population density within each class, which may not hold true across different parts of the source zone. They suggest adopting more objective approaches that determine weights based on the specific characteristics of the area under consideration.

Intelligent dasymetric mapping

Intelligent dasymetric mapping method was introduced by Mennis & Hultgren (2006a) as a solution to subjective weights in weighting layer. In contrast to multi-class dasymetric mapping, this method estimates weights by sampling population densities associated with ancillary spatial classes in the source zone. Alternatively, analysts can assign preset densities to certain ancillary classes based on local knowledge, for example, setting the density for water bodies to zero.

Sampling means to picking source zones (e.g. census blocks) that are linked to a specific ancillary class (e.g., specific type of land cover), i.e. that are indicative or representative for this ancillary data. As proposed by (Mennis & Hultgren, 2006b), three main types of sampling are possible: *centroid*, *contained* or *percentage cover* approach. Centroid approach selects only those source zones (e.g. census blocks) whose centroid falls within individual ancillary class (e.g., specific type of land cover). Contained approach takes source zones that are wholly contained within an individual ancillary class while percentage cover uses a threshold percentage (e.g., 70%) to select source zones whose area of occupation by a single ancillary class is greater than or equal to this threshold. Once the source zones are linked to ancillary classes based on sampling, their population counts and areas are used to calculate population densities. Finally, these population densities are used as weights to reallocate population from source zones to target zones (Mennis & Hultgren, 2006a). If the parameters are set appropriately, intelligent dasymetric mapping can serve better quality results than areal weighting or binary dasymetric (Mennis & Hultgren, 2006b).

The concept of deriving weights from ancillary data through self-training has evolved over the years into more comprehensive and computationally intensive disaggregation methods. These methods often incorporate complex spatial statistics to model the correlation between ancillary data and source zone population data. Sapena et al. (2022) classify these methods as statistical methods.

Statistical areal interpolation methods

Statistical areal interpolation methods utilize ancillary data in statistical functions to establish functional models capable of describing relationship between spatial distribution of ancillary data and spatial distribution of population data in source zone that is to be reallocated (Comber & Zeng, 2019). This method makes assumption of correlation between ancillary data and source zone data which is why this should be verified before running statistical analysis (Qiu et al., 2022). Main approach of any method relies on use of statistical techniques like least squares, geographic weighted regression, linear regression or random forest to model relationship between source data and ancillary data (Monteiro et al., 2021; Sapena et al., 2022; Song et al., 2019). The resulting model is then used as weighting layer and applied to target zones to estimate final population values based on their attributes. Additionally, Comber & Zeng (2019) highlight geostatistical methods as an emerging subgroup within this category. Although originally used for point interpolation, geostatistical methods can include spatial autocorrelation in the modelling process and inherently maintain volume-preserving (pycnophylactic) properties (Comber & Zeng, 2019).

Statistical methods tend to perform well when remotely sensed ancillary data are combined with other geospatial datasets, such as road networks (Sapena et al., 2022). Comber & Zeng (2019) attribute this success to geodata acting as detailed control variables that enable more precise population redistribution. This insight has paved the way for hybrid methods, which integrate the strengths of dasymetric mapping with the analytical power of statistical approaches to produce more accurate disaggregation results.

Hybrid methods

Hybrid methods can be viewed as an advanced form of intelligent dasymetric mapping, where the weighting layer is derived from a statistical model as used in statistical methods. The key assumption here is that these weights represent population density only within populated areas, while unpopulated regions are excluded from the redistribution process (Wang & Wang, 2024). Compared to other approaches, hybrid methods generally yield more accurate results but require more complex modelling and higher computational resources (Sapena et al., 2022).

2.2.4 Review on Spatial Disaggregation Methods

Chapters 2.2.2 and 2.2.3 describe the theoretical foundations of spatial disaggregation of population data; however, these form only the basis for methodologies applied in practice. No single disaggregation method performs optimally in all scenarios, and their universal applicability is often constrained by the availability and diversity of suitable geospatial data (Swanwick et al., 2022). Therefore, this chapter aims to provide an overview of the various practical disaggregation approaches and highlight the key concepts that are common across different methods.

Methodologies adopted in practice

The scientific community remains actively engaged in developing new and improved methodologies for spatial disaggregation, as reflected in the growing number of publications on this topic. All these studies aim to produce higher-quality disaggregated population data, pushing the boundaries of how ancillary data is utilized within different methodologies. For instance, Baynes et al. (2022), Calka et al. (2016), Cartagena-Colón et al. (2022), Karunarathne & Lee (2019), Liu et al. (2023), Mennis & Hultgren (2005), Monteiro et al. (2018, 2019, 2021), Pajares et al. (2021), Reiter et al. (2023), Stevens et al. (2015), Swanwick et al. (2022) all employ approaches based on dasymetric mapping, differing primarily in the types of ancillary data used and how it is incorporated into the process.

For example, Mennis & Hultgren (2005) rely solely on land cover data for population redistribution using multi-class dasymetric mapping, while Baynes et al. (2022) extend this by including land use data to inform weighting for population reallocation. Monteiro et al. (2018) integrate nighttime lights, land cover, and road networks within linear and generalized additive regression model that combines dasymetric mapping and pycnophylactic interpolation. Similarly, Monteiro et al. (2019) employ land cover, elevation, terrain development, and distance to water bodies using ensembles of decision trees, also blending dasymetric mapping with pycnophylactic interpolation. Stevens et al. (2015) utilize over 20 types of ancillary data within a random forest model to determine weights for dasymetric mapping, whereas Reiter et al. (2023) base their approach on

detailed building-specific information, such as volume and footprints, to guide the dasymetric mapping process.

From this sample of diverse scientific articles, it is evident that the disaggregation process is not a straightforward, one-size-fits-all approach. Instead, it manifests in various forms depending on the chosen base model and the types of data employed. Therefore, spatial disaggregation should be viewed as a flexible framework that offers minimal prescriptive guidance, allowing for adaptation based on specific data availability and methodological preferences.

Shared Conceptualizations Across Disaggregation Methods

Spatial disaggregation methods are used to redistribute aggregated population data from arbitrary spatial units, such as administrative zones, into spatial units that better represent the actual distribution of the population. As highlighted by Mennis & Hultgren (2006a) disaggregation is a process which means it has well established general procedures, and no matter the approach, all methods share some same characteristics. For example, all the methods include administrative or statistical units, often in vector representation, and information on their population counts. Furthermore, all the methods are oriented in the same direction and start from source zone with aggregated data and result in target zones with disaggregated data. Even though these zones may be different in shape and size, they have uniform meaning throughout the methods. Moreover, all methods rely on additional parameters that guide this process. These parameters can range from simple area proportion, subjectively estimated population density for a specific area to more complex statistical models that consider correlation of different georeferenced data with existence of population. But no matter the way parameters are gained, they serve as weights in final disaggregation step. From a conceptual standpoint, the primary distinction between methods lies in the use of ancillary data: some rely solely on source zone characteristics, while others incorporate external data. Additionally, representation of ancillary data may also be the distinction here as some methods use geodata in vector structure while others consider raster data or a combination of both as source of information.

2.2.5 Practical Implementation of Spatial Population Disaggregation

Research in the field of spatial disaggregation can be viewed as twofold. On one hand, some authors focus on developing new and improved disaggregation techniques, as discussed in the previous chapter. On the other hand, there is significant effort dedicated to creating practical disaggregation tools. In the literature, these tools are generally divided into standalone solutions, such as custom-coded scripts, and extensions to existing platforms, including programming language libraries or GIS toolboxes.

Standalone solutions are often developed as proof-of-concept implementations for newly proposed disaggregation methods and typically exist as independent scripts. For example, Swanwick et al. (2022) provide code for their refined dasymetric mapping approach written in programming language for statistical computing, R. Similarly, Monteiro et al. (2018, 2019), Stevens et al. (2015) have extended existing R source codes to accommodate their novel disaggregation techniques. Another example is Monteiro et al., (2021) who implemented a new method using Python scripts, leveraging machine learning libraries such as scikit-learn and TensorFlow.

In contrast to standalone solutions, extensions to existing technologies aim to promote broader adoption of spatial disaggregation methods by implementing well-established approaches. Batsaris & Zafeirelli (2023) note that only a limited number of such tools are currently available, which may limit the wider use of disaggregation techniques in the community. In the domain of programming languages, examples can be found in R packages *pycno* and *dissever* (see Monteiro et al., 2018). Dissever package implements Dissever algorithm, hybrid method that combines pycnophylactic interpolation with regression-based dasymetric mapping, to spatially redistribute population data (Moreira Ribeiro, 2017). Meanwhile, pycno package, developed by Cris Brunsdon, supports pycophylactic interpolation based on the algorithm introduced by Tobler (1979).

In addition to scripting packages and libraries, extensions to GIS also fall within this category. A well-known example is Dasymetric-Mapping Extension (DME) developed for ESRI ArcGIS which applies areal interpolation to facilitate multi-class dasymetric mapping (Sleeter & Gould, 2008). Similarly, Qiu et al. (2012) developed an ArcGIS extension offering four disaggregation methods: areal weighting, pycnophylactic

interpolation, binary dasymetric mapping, and 3-class regression dasymetric mapping (Qiu et al., 2012). More recently, Flasse et al. (2021) introduced a GRASS GIS extension that utilizes machine learning-based dasymetric mapping. This tool calculates weighting layers by applying machine learning techniques over land cover and land use datasets (Flasse et al., 2021).

All the previously described approaches have limitations that hinder their broader adoption. On one hand, scripting solutions require advanced programming skills, while on the other hand, GIS extensions are often not open-source, typically require proprietary software licenses, and are rarely available as ready-to-use, off-the-shelf tools (Batsaris & Zafeirelli, 2023). To address these challenges, Batsaris & Zafeirelli (2023) proposed a web-based tool for population reallocation called the PoD tool. This tool integrates four disaggregation methods, areal weighting, volume weighting, binary dasymetric mapping, and float dasymetric interpolation, offering a user-friendly, ready-to-use application that produces disaggregated population data with minimal user input.

Given this review of current practical approaches to population disaggregation, it is clear that a certain level of data processing expertise is required to perform any form of disaggregation. As the importance of population data in geospatial applications continues to grow, Batsaris & Zafeirelli (2023) emphasize the need for automated solutions that require minimal user input. According to the authors, such solutions eliminate the complex stages of data collection and preprocessing, making disaggregation more accessible and better suited to meet the needs of both experts and novices alike.

2.3 Time Consideration in Population Distribution

As explained in Section 2.1.2, population data refers to the number of people residing within a given area. However, due to constant population migrations, this data is inherently time-dependent, making the temporal reference crucial for comprehensive interpretation and application. Official statistics, which are the most common source of population data, typically provide snapshots in time, often with limited temporal resolution. Depending on the data collection approach employed in the creation of official statistics, temporal resolutions can vary from low to high. For instance, censuses, due to their complexity, are usually conducted every ten years and often take several days or

weeks to enumerate the entire population (Nebiler & Neupert, 2020). This puts constraints on practical usability of data as, firstly, data is available every ten years making these decennial snapshots uninformed of population changes within this period, and secondly, data analysis can take up to few years making data obsolete before even released (National Research Council, 2007). Limitations of censuses are usually bridged with sample surveys. As these are less expensive to conduct, sample surveys are more temporally frequent (monthly, quarterly or annually), undertaken periodically or ad hoc (United Nations Department of Economic and Social Affairs, 2008). While ad hoc surveys can satisfy immediate needs, e.g. interdecennial estimation, they cannot set a framework for continuous time series. On the other side, periodical surveys act with cyclic repetition and can monitor observed phenomena over a period of time. Often, sample survey approach is used to provide insight into temporal changes of population in between the censuses (United Nations Department of Economic and Social Affairs, 2005). Lastly, administrative records, drawn from official registers, have the potential to provide population data with almost daily temporal resolution. But this is dependent on the maintenance of the registers and may not be reliable in cases of low-frequency updates (United Nations Economic Commission for Europe, 2007).

Goodchild (2013) argues that time and space in geography are inseparable and must be modelled jointly. Since most available population datasets are based on census data with a fixed census time reference (Wang & Wang, 2024), fulfilling the needs of many applications requires not only spatial disaggregation but also temporal adjustment.

2.3.1 Time modelling in GIS

Pred (1977) defines time geography as an approach focused on understanding human activities within the context of space and time. It is constrains-oriented and recognises every activity occurs at a specific location and lasts for a given period (Miller, 2003). In GIS this approach is referred to as people-based approach, as it tracks entities, i.e., individuals, in their movements through time and space. However, Miller (2007) distinguishes a second approach for modelling time in GIS, known as place-based approach, which focuses on changes occurring at fixed spatial locations over time. According to King (2019), it is this place-based approach, which monitors changes from

a spatial perspective, that is more commonly used to handle time in GIS rather than the people-based approach.

Geospatial data is generally understood to have three main components: space, time, and attributes. Each of these components can change, meaning that both spatial locations and attribute values may vary over time. Based on this, Yuan (1997) defines six main types of spatial/temporal changes that make use of fixed, controlling and measuring component. These types are divided primarily based on the fixed component, while the remaining two components are used to monitor the change. For example, if the attributes of a given area change over time or across space, then space is fixed, time acts as the controlling component, and attributes are the measured variables (Yuan, 1996). To model these types, the literature proposes numerous temporal GIS models, which can be classified in various ways (see Siabato et al., 2019). Yuan (1996) offers a straightforward classification based on how time is handled in the model, distinguishing between timestamping spatial objects and modelling time through events or processes.

Timestamping of spatial objects

The simplest example of the timestamping approach is the Snapshot Model (Table 2.1), where the temporal aspect of data is represented by a series of layers of the same thematic data, each tagged with a different timestamp (Armstrong, 1988). Examples include shapefiles containing land use polygons or remotely sensed images taken at different times. Although these snapshots are arranged sequentially, they are not temporally linked, and thus do not inherently capture or explain changes occurring between consecutive timestamps (Siabato et al., 2019). This means that to detect changes, the layers must be directly compared within a GIS environment. There have been attempts to improve Snapshot Model. Beller et al. (1991) introduced the Temporal Map Sets Model with the core concept of system being capable of making interpolation between the layers and creating a layer whose cells did or did not participate in the change – binary temporal map sets. While the Snapshot Model is straightforward to implement in GIS systems (King, 2019), its main drawbacks are data redundancy and the risk of data inconsistency (Yuan, 1996).

Table 2.1
Temporal shapshots of Zagreb administrative unit (modified from Hedefalk et al., 2014)

ID	Name	timeStamp	Geometry
15	Kaptol	14.3.1813	*(polygon 1)*
22	Zagreb	7.9.1850	*(polygon 2)*
89	Zagreb	9.11.1990	*(polygon 2a)*

The second type of timestamping models focuses on representing the evolution of spatial objects by capturing their changes over time. As King (2019) explains, this can be done in multiple ways. One example is the Space-time Composites Model, also considered a refined version of the Snapshot Model, proposed by Langran & Chrisman (1988). This model represents conceptual changes of spatial objects at discrete points in time by storing these changes within layers of time composites (Yuan, 1996). Unlike the simple Snapshot Model, each composite maintains an attribute history that describes the composite from which it was derived (King, 2019). However, despite this improvement, the model is still limited in its ability to track individual objects and detect changes over time (Hedefalk et al., 2014). Another approach, introduced by Worboys and Duckham (2004), records every change in a spatial object by adding it as a temporal dimension known as the object's lifeline (King, 2019). This method enables modelling the entire lifecycle of an object, with timestamps for each significant event such as creation, spatial modifications, or typological changes (Table 2.2) (Hedefalk et al., 2014).

Table 2.2
Zagreb administrative unit stored as object lifelines (modified from Hedefalk et al., 2014). startData and endDate represent validity period for the administrative unit

ID	AdminUnitID	Name	startDate	endDate	Geometry
2	Au_Zagreb_1	Kaptol	14.3.1813	7.9.1850	*(polygon 1)*
3	Au_Zagreb_2	Zagreb	7.9.1850	9.11.1990	*(polygon 2)*
4	Au_Zagreb_3	Zagreb	9.11.1990	-	*(polygon 2a)*

Another approach within this category is the Spatiotemporal Object Model proposed by Worboys (1992). In this model, real-world objects are represented as spatiotemporal atoms in a three-dimensional space, where time serves as the third dimension (Figure 2.6).

A key characteristic of this approach is that spatiotemporal atoms are homogeneous units with properties fixed in both time and space, while the temporal dimension allows the model to capture changes in spatial attributes over time (Siabato et al., 2019). Unlike the Snapshot Model, the Spatiotemporal Object Model represents only sudden changes occurring at discrete points in a linear temporal sequence (Yuan, 1996). Approaches like this, where object is considered to be unambiguously identified makes these types of models classified as object-oriented models. According to Siabato et al. (2019) object-oriented models provide an intuitive understanding of the dynamic behaviour of independent objects, enabling effective tracking of their changes. This characteristic has made object-oriented models widely accepted and foundational for many adopted spatiotemporal modelling approaches (Siabato et al., 2019).

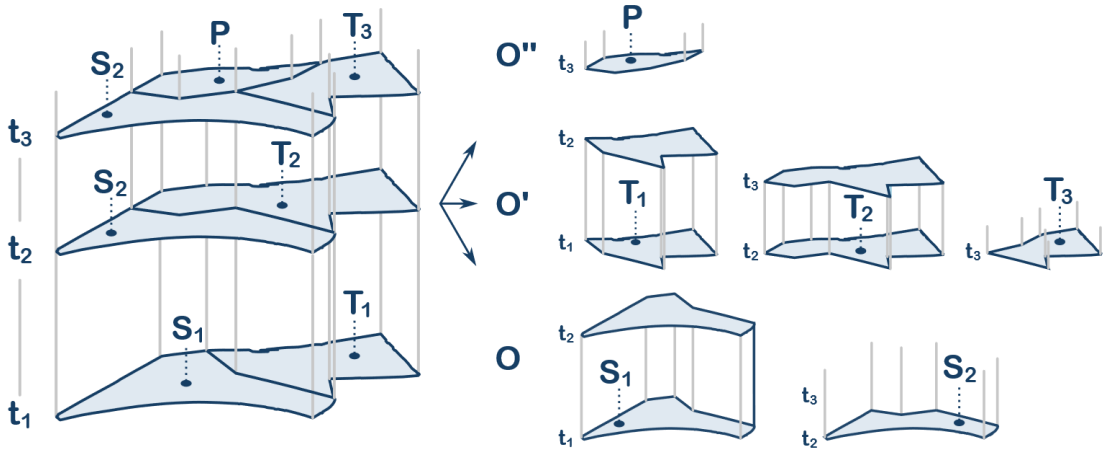


Figure 2.6
Spatiotemporal objects as spatiotemporal atoms (modified from Worboys, 1992)

A main disadvantage of timestamping approaches is their limited ability to provide detailed information about the nature of the change itself; instead, these models primarily focus on the occurrence of a change event in the observed phenomenon.

Spatiotemporal information as event or process

Dynamic behaviour of phenomena can be modelled in a way that it enables informative spatiotemporal queries that address not only *where* and *when*, but also *why* and *how fast* changes occurred. Such approaches describe complex spatial, temporal, and thematic interrelations, allowing the identification of patterns, trends, and causes of change within

the space-time environment (Langran, 1992 according to King, 2019). King (2019) refers to these as spatiotemporal models, defining them as integrated representations where events, actions, and processes are explicitly modelled alongside the objects they modify.

Siabato et al. (2019) provide an overview of several empirical implementations of this approach. For example, Three-Domain Model proposed by Yuan (1996) defines spatial (point, lines, polygons, cells), temporal (time instants and periods) and thematical (aspatial and atemporal attributes) objects in three separate domains and links them to create dynamic representation of geography (King, 2019). Unlike Snapshot Models, where time is tied to specific locations, this model treats time independently, offering flexibility to represent reality and monitor various changes, including attribute modifications and both static and dynamic spatial transformations (Siabato et al., 2019).

Another example is the Process-based Spatiotemporal Model introduced by Yang & Claramunt (2003). This approach considers entities, processes and changes as modelling primitives and describes them separately within spatial, temporal, and thematic domains interconnected by domain links (Siabato et al., 2019); similarly to approach in Yuan (1996).

Furthermore, the Event Oriented Spatiotemporal Data Model (ESTDM) developed by Peuquet & Duan (1995) organizes event-caused locational changes in a timestamped raster layers. Starting with an initial state (the so-called base map), ESTDM stores changes in state over time rather than storing complete instances (Peuquet & Duan, 1995) (Figure 2.7). This reduces data redundancy and facilitates easier tracking of changes across space, time, and theme-specific values (Siabato et al., 2019). A potential limitation of this model is its reliance on raster-based representation, which may restrict flexibility (King, 2019).

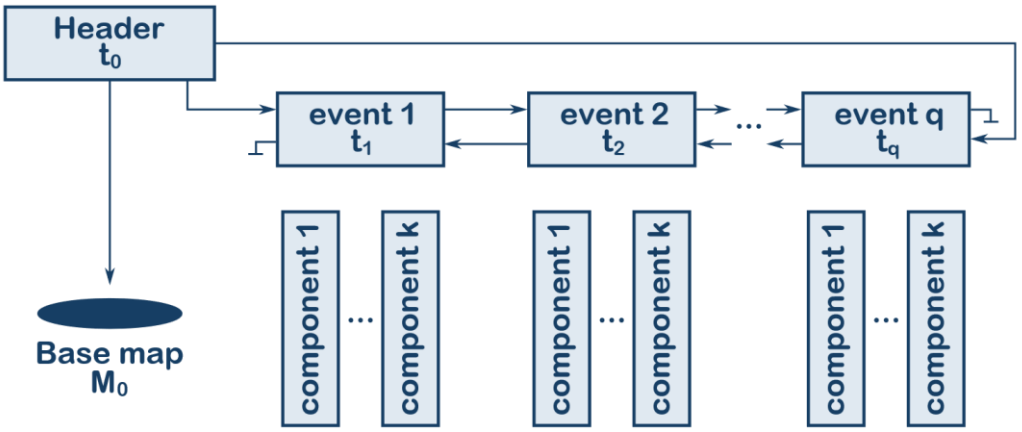


Figure 2.7
Eventu Oriented SpatioTemporal Data Model (ESTDM) main elements and pointer structure (modified from Peuquet & Duan, 1995)

Each of the models described above has its own advantages and limitations. For example, simple approaches like the Snapshot Model are easy to implement in GIS but cannot capture complex spatiotemporal relationships. In contrast, spatiotemporal models enable more in-depth analysis by explicitly representing events, processes, and interactions, but they require more complex modelling and often depend on diverse data types that may not always be available. Therefore, the choice of model should be guided by the specific goals and requirements of the intended model's application (Renner et al., 2018).

2.3.2 Intercensal population data

Although spatiotemporal population modelling is highly valuable in many areas of application, for example in risk assessment that requires having detailed information about real-time population distribution (Renner et al., 2018), i.e. knowing who is where, it may not always be necessary or feasible. Ch et al. (2021) argue that place of residence, referred to as nighttime population, is often key demographic data used for urban planning, infrastructure design, housing and health services organization, as well as for making temporal comparisons of population size and change. Furthermore, Wang & Wang (2024) indicate that production of time accurate population distribution relies on high temporal resolution data such as mobile phone locations. While access to such data may be a major barrier to widespread use of spatiotemporal disaggregation models, this highlights that timestamping approaches still offer a practical and suitable solution for many real-world applications.

Neal et al. (2022) state that censuses are golden standard for population data; however, their decennial resolution is insufficient to capture changes occurring between census years. This limitation has driven the development of several temporally refined population data products in recent years, e.g. WorldPop 100m resolution annual population grids, highlighting the growing importance of intercensal population estimation (Neal et al., 2022).

In their work, Silva et al. (2011) produce improved annual population growth rates to temporally disaggregate available census data. The authors first interpolate decennial cumulative growth rate into annual growth rates using vital statistics data (births and deaths rates). Vital statistics are available for every year in the intercensal period and are used to rectify interpolated growth rates. In the second step, time-series model is used to distribute population changes over the years based on the improved annual population growth rates. Because the annual growth rates are constrained so that their cumulative sum matches the decennial census totals, the method maintains consistency both at the annual and decadal scale.

Creation of WorldPop annual population grids follows a similar approach. First, data on population counts per administrative units are taken from the global database. If needed, these population counts are projected to reference year using estimated population densities. Following this, the data are spatially disaggregated to achieve higher resolution using dasymetric mapping, with weights determined through a random forest method (World Pop, 2024).

Wang et al. (2020) created annual intercensal population estimates for age, sex and race/ethnicity subgroups at the level of census block. Authors firstly considered boundary change and redistributed more recent population counts onto boundaries from previous census year using area intersection ratio. Then, estimation of population for a certain year was calculated by weighting population with its temporal proximity to both referent censuses (Wang et al., 2020).

Han & Howe (2024) propose neural network model designed to disaggregate spatially and temporally aggregated data into finer spatial and temporal resolutions. Even though disaggregation is an integrated process, it can be viewed as sequential in its execution. The

model is based on a Structurally-Aware Recurrent Network, which begins with discrete points in time, such as census counts, and applies statistical interpolation to estimate finer temporal series by leveraging temporal dependencies across these discrete time points within each spatial unit. Temporally refined counts are then spatially reallocated into smaller areas using spatial disaggregation methods.

From the reviewed articles, it is seen that usually one layer of spatial units holds the anchor value (e.g. census data) while changes in the value are provided in timestamped layers with reference to these spatial units. Often, temporal difference between these timestamps (layers) is used to interpolate population counts. Neal et al. (2022) build this on and state that administrative records on population change (migration, birth, death) and sample household surveys, can be useful for these purposes as these are collected with higher accuracy or more frequently. Keeping in mind that intercensal nighttime population supports a variety of applications, the interpolation approach between censuses will be used, primarily for modelling simplicity, as the basis for temporal disaggregation in the proposed population model in this dissertation.

Ontology and Semantic Web

The primary focus of this research is the development of an ontology model. Accordingly, this chapter examines the conceptual foundations underpinning the creation of an ontology model for the spatiotemporal disaggregation of population data within a semantic web environment. To ensure a comprehensive understanding of the topic, the chapter introduces key concepts relevant to the domain and explores their interrelationships, discusses methods for the creation and representation of data in the semantic web, and reviews existing approaches to the ontological modelling of processes.

3.1 Concept of Semantic Web

The term semantic pertains to the meaning of words or the processes through which meaning is derived. While humans can easily deduce and infer meaning from structured, semi-structured, or even unstructured but contextually related information, such as in everyday communication, computers encounter considerable difficulty in this regard (Frontiersi, 2018). Consequently, computers face challenges in advancing along the Data, Information, Knowledge, Wisdom (DIKW) pyramid; that is, in transforming raw data into meaningful information, generating knowledge from that information, and ultimately applying that knowledge to produce wisdom (see Baškarada & Koronios, 2013).

Tim Berners-Lee borrowed the term *semantic* and used it to denote his vision of a new form of web – Semantic Web, in which content is provided in formats that are understandable not only to humans but also to machines. In contrast to traditional web, where meaning is locked in HTML documents written in natural, human-readable language, the Semantic Web is based on the premise that every extract of information on the web is given semantic description, such that machines can infer about what it is and use it accordingly (Hogan, 2020). To illustrate this distinction, Baučić (2014) considers Semantic Web as a database which can be searched and interpreted by the computers, in contrast to the current web that can be seen as a book only people can search and interpret.

This new conceptualization of the web is not intended to replace the web as we know it but should be seen as an extension to better exploit its potentials (Berners-Lee et al., 2001). According to Baučić (2014), the Semantic Web enables more effective utilization of the current web by introducing semantic information that facilitates improved search,

linking, integration, and retrieval of data. Furthermore, automatic reasoning applied to semantically enriched data can be leveraged to support applications such as the automation of tasks within the web services lifecycle (De Souza Neto et al., 2018) and in data-analytical processes (Bednár et al., 2024).

Tim Berners-Lee conceptualized the Semantic Web architecture as a layered framework of standards and technologies that incorporate formal, machine-readable semantics into data, thereby enabling the creation of data stores, supporting vocabularies, and rules for data processing (Berners-Lee et al., 2001). This framework, commonly referred to as Semantic Web technology stack (Figure 3.1), describes how technologies are layered to support this new form of web but also how the new web is built on top of the traditional web technologies such as URIs, Unicode and XML. To ensure a stable foundation for the implementation and future advancement of the Semantic Web, the World Wide Web Consortium (W3C) plays an active role in the development and standardization of the Semantic Web technology stack.

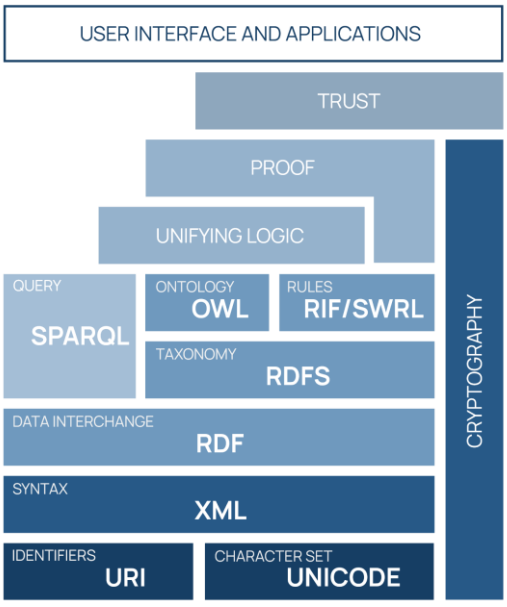


Figure 3.1
Semantic Web technology stack (modified from Frontiersi, 2018)

As previously discussed, the Semantic Web is founded on the concept of semantically interconnected data that can be interpreted and reasoned about by machines. This places ontologies, representations of domain-specific knowledge, at the core of the Semantic

Web paradigm. However, to effectively utilize the semantics embedded within ontologies, several complementary technologies for their creation and application are required. For instance, the Resource Description Framework (RDF) is used to describe and structure the data, while SPARQL enables querying across the global network of interconnected data. The Web Ontology Language (OWL) provides formal semantic representation, and the Rule Interchange Format (RIF) and Semantic Web Rule Language (SWRL) are used to define additional rules on data semantics. The upper layers of the architecture, Unifying logic, Proof, Trust and Cryptography serve to ensure that the Semantic Web remains reliable, secure, and trustworthy for real-world applications (Al-Feel et al., 2008).

Given that the primary goal of this research is the development of an ontology model, the subsequent discussion will concentrate on the technologies pertinent to capturing semantics, as well as representing and querying data.

3.2 Foundations of Semantic Web

The Semantic Web builds on a simple but powerful data model: representing knowledge as triples of subject–predicate–object. The creation of meaningful and interoperable triples, however, requires stable conceptual foundations grounded in ontologies and supporting technologies that define shared meaning, clarity, and consistency across systems. These foundations establish a shared layer of data that can be exchanged and reused across diverse contexts.

3.2.1 Ontology

Ontology is defined as philosophic field concerned with the nature of existence, seeking to explain what entities exist and how they relate to one another (Maedche, 2002). Within the context of information sciences, ontologies are used to represent knowledge about a partition of reality in an intelligent computer system and are described as a "formal specification of a shared conceptualization" in a domain of interest (Gruber, 1993). This definition by Gruber (1993) emphasizes two main concepts: formal specification and shared conceptualization. While shared conceptualization means that the real world is too complex for representation and that it should be simplified to a formal, agreed-upon model of a domain for representation in a system, formal specification denotes need for codifying this shared conceptualization in a logical language with well-defined syntax and semantics

(Neuhaus, 2018). This approach enables both humans and machines to share a common understanding of the entities that exist within a domain, their properties, and the relationships among them.

To represent knowledge, ontologies in a domain of interest conceptualize the real world using classes, properties (attributes), individuals and relations (National Academy of Sciences Engineering and Medicine, 2022). Classes and their subtypes (subclasses) constitute abstract groupings of entities, typically organized into hierarchical structures (taxonomies) and characterized by their properties. Individuals are concrete members of classes, or class instances, and relations define connections between classes, between classes and instances, or among instances themselves. This structured set of concepts enables the creation of a human-readable vocabulary and taxonomy of representation terms, ensuring a coherent and semantically consistent knowledge representation. But, to ensure their constrained interpretation and use by the computer, i.e. to support automated processing, ontologies rely on formal axioms applied to classes and relations (Hogan, 2020). These axioms augment the semantic framework of the vocabulary with formal logic and constraints, which are essential for automated reasoning and inference.

The demand for knowledge representation across diverse application domains has led to the development of a wide range of ontologies. These ontologies vary in the types of knowledge they convey, and Kharbat & El-Ghalayini (2008) classify them along three dimensions: level of formality, level of generality, and types of primitives.

The level of formality distinguishes ontologies according to the degree of expressivity and precision in concept specification (Figure 3.2). Uschold & Gruninger (1996) propose that, based on the formality of knowledge representation, ontologies can be categorized as informal, formal, or semiformal. Informal ontologies are typically expressed in natural language and primarily serve to convey the hierarchy of concepts within a domain, identifying relevant concepts and their interrelationships. In contrast, formal ontologies are designed for computational use, employing complete axioms to formally define semantics within a model. Semiformal ontologies occupy an intermediate position, incorporating axioms into the hierarchy but expressing only basic, atomic statements rather than complex logical expressions (Uschold & Gruninger, 1996). Corcho et al.

(2003) further classify ontologies based on the restrictiveness of their semantic models into lightweight and heavyweight ontologies (Figure 3.2). Lightweight ontologies primarily define taxonomies, identifying concepts, relations, and properties, often supplemented with a minimal set of axioms, such as subclass or subproperty declarations, to capture basic semantics. Heavyweight ontologies extend lightweight ontologies by incorporating additional axioms and constraints, enhancing their reasoning and inferential capabilities. While heavyweight ontologies provide greater semantic precision and reduce ambiguity, their development is more resource-intensive and time-consuming (Baučić, 2014).

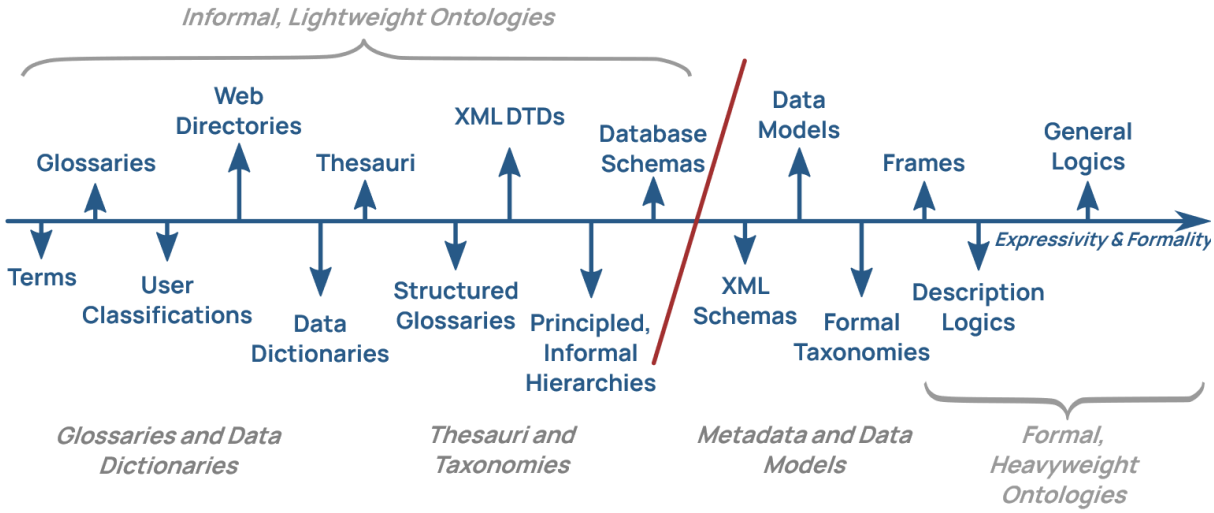


Figure 3.2
Spectrum of ontology kinds based on expressivity and formality in concept specification (modified from Wong et al., 2012)

Ontologies are developed for specific domains of interest and are not generic or isolated; rather, they are shaped by particular user perspectives and their interactions with other domains. Consequently, a single domain can be represented from multiple perspectives, resulting in different descriptions of the same concepts. Within the context of the Semantic Web, these multiple descriptions are not problematic, as the framework operates under the open-world assumption, wherein knowledge is considered unbounded and continually updated with new information. The Semantic Web therefore relies on multiple ontologies to construct its knowledge base and is designed to facilitate the reconciliation of different ontologies into a unified knowledge graph. To support this process, it employs ontologies at varying levels of conceptualization.

Guarino (1998) classifies ontologies according to the level of detail in their conceptualization, distinguishing three levels of granularity: top-level (general) ontologies, domain and task ontologies, and application ontologies (Figure 3.3).

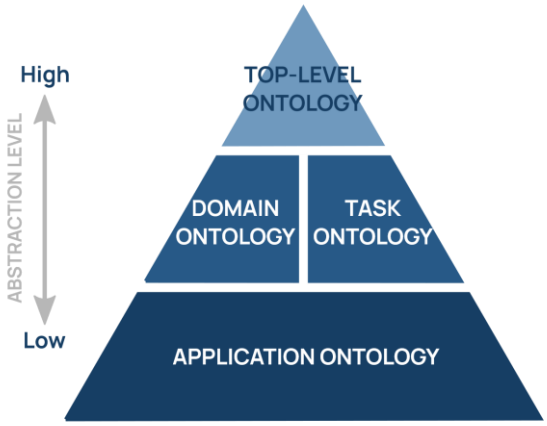


Figure 3.3
Kinds of ontologies in general ontology classifications (modified from Mohamad et al., 2021 and Polenghi et al., 2022)

Top-level ontologies describe general concepts such as space, time, or objects, which are universal and independent of any specific domain or problem. Several such ontologies exist and some of widely used are SUMO (Suggested Upper Merged Ontology), BFO (Basic Formal Ontology) and DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering). DOLCE adopts a descriptive approach, representing the world from the perspective of human perception and conceptualization. It distinguishes between endurants (entities that persist through time), perdurants (entities that unfold over time), qualities (properties inherent in entities), and abstracts (non-physical entities, such as numbers), reflecting a perspective closely aligned with natural human descriptions (Borgo et al., 2022). BFO, in contrast, originates from biomedical sciences and is realist in orientation, representing entities that exist independently of human conceptualization while emphasizing scientific rigor and interoperability. It establishes a fundamental distinction between continuants and occurrents, which serves as a foundation for building additional ontologies (Arp et al., 2015). Finally, SUMO aims for comprehensive coverage, integrating insights from multiple existing ontologies and defining broad concepts such as Object, Process, and Relation (Niles & Pease, 2001).

Domain and task ontologies encompass concepts that are broadly applicable across general domains (e.g., topography) or activities (e.g., diagnosing). They further specialize the general concepts defined in top-level ontologies to facilitate knowledge integration within a specific domain (Guarino, 1998). Examples of domain ontologies include OGC's GeoSPARQL vocabulary, which represents geospatial objects; W3C's RDF Data Cube vocabulary, for statistical data representation; and W3C's OWL-Time ontology, which describes temporal concepts. In contrast to domain ontologies, which primarily define static concepts, task ontologies are designed to represent knowledge about particular tasks or activities, with the aim of achieving reusability across different domains. An illustrative example is the Function Ontology, which describes functions and their execution independently of the underlying technology.

Application ontologies integrate concepts from both domain and task ontologies, often specializing them to address specific application requirements. Within these ontologies, concepts typically assume the roles of domain entities while executing activities defined by task ontology concepts (Guarino, 1998).

Finally, representation ontologies provide meta-level vocabularies that serve as foundational building blocks, such as classes, properties, and individuals, for describing other ontologies (Kharbat & El-Ghalayini, 2008). rominent examples include RDF Schema and the Web Ontology Language (OWL), both of which are integral components of the core Semantic Web technology stack.

The final classification of ontologies is based on the types of primitives employed to model the real world. Jurisica et al. (2004) differentiate among three categories: static ontologies, which represent the world as a fixed set of concepts, their attributes, and relations; dynamic ontologies, which capture the evolving aspects of the world, such as states or transitions; and social ontologies, which model social constructs, including actors, positions, and commitments.

Semantics captured in ontologies form the very core of the semantic web, but for this knowledge to be exploited, technological infrastructure must be in place.

3.2.2 Resource Description Framework (RDF)

Data on the web originates from diverse sources and is structured according to various abstract models, for example, CSV represents tabular data, while XML represents hierarchical trees (Hogan, 2020). Such heterogeneous data structures pose a challenge to the vision of the Semantic Web, in which distributed data should be seamlessly integrated into a global network. This challenge prompted the development of a unified data model that is sufficiently generic to represent different types of data, thereby facilitating interoperability, integration, and data exchange.

The Resource Description Framework (RDF) is a standard framework for describing resources, defined as anything of interest, within the Semantic Web. RDF is based on the concept of triples, comprising subject–predicate–object statements that represent relationships between resources, thereby modelling the web as a directed graph. This structure closely resembles natural language expressions, making the model sufficiently flexible to capture data from diverse sources. For instance, when converting tabular data into RDF triples, each row is treated as a subject, each column as a predicate, and the corresponding cell value as the object of RDF triple (Table 3.1).

Table 3.1
Example tabular data for RDF triple creation

	Population	Country
Gvozd	2047	CRO

Following the proposed conversion rule, tabular data from Table 3.1 can be interpreted as follows:

Triple no. 1: Gvozd is subject, hasPopulation is predicate, 2047 is object, and

Triple no. 2: Gvozd is subject, withinCountry is predicate and CRO is object.

Given in the syntax of RDF triples, this information is represented as (Example 3.1):

Example 3.1

(Gvozd, hasPopulation, 2047), and

(Gvozd, withinCountry, CRO),

or visualized in directed graphs like (Figure 3.4):

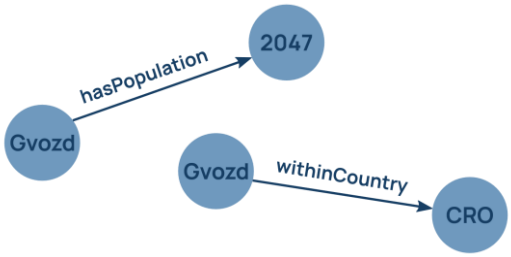


Figure 3.4
Tabular data representation in RDF triples

Information on the web is often distributed across multiple RDF graphs, with different pieces of the same knowledge located in separate graphs. To obtain a complete view, it is necessary to merge identical nodes into a single directed graph (Figure 3.5). By linking related resources in this manner, the data become linked data, thereby extending the informational context within the global semantic network. To enable this linking, nodes and edges in RDF triples must be uniquely identifiable and distinguishable from other resources on the web. Uniform Resource Identifiers (URIs), which are widely used in the traditional web, provide a suitable mechanism for this purpose.

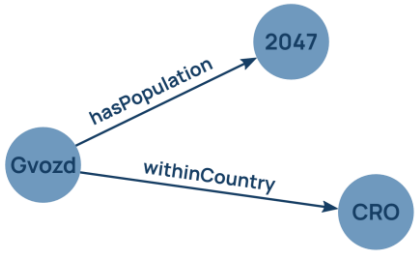


Figure 3.5
Single directed graph of complementary RDF triples

3.2.3 Uniform Resource Identifiers (URIs)

Uniform Resource Identifiers (URIs) are employed to uniquely identify every resource within RDF triples in the global semantic network. A resource can represent any entity, ranging from tangible objects (e.g., a book) to abstract concepts (e.g., a polygon). The key principle is that each segment of information is assigned a unique identifier, enabling precise inference about the resource being referenced (Belozerov & Klimov, 2022). For example, a URI for the concept of *population* must clearly indicate whether it pertains to its meaning in demographic studies or in biology.

Although URIs are designed to be globally unique, the same concept can be represented multiple times on the network, resulting in multiple distinct identifiers. Baučić (2014) identifies this as a primary challenge of the Semantic Web and outlines three potential solutions. The first approach involves manually asserting that all such URIs refer to the same concept. The second approach relies on predefined URIs for all possible concepts, which are then consistently applied. The third approach establishes consensus among relevant stakeholders regarding the URIs assigned to specific concepts. In practice, the most common strategy is to reuse existing knowledge rather than create new identifiers, a topic that will be explored further in subsequent chapters.

URIs follow a specific syntax, which Hogan (2020) describes as comprising multiple components. In common usage, a URI typically consists of two main parts: (1) the URI scheme, which specifies the protocol for accessing the resource, such as http, and (2) the textual description of the resource being referenced, which includes the host (the server location, often represented as a domain name), the path (the location of a file), and the fragment (a reference to a specific piece of information within the file). This structure can be illustrated with a simple example, such as the URI for the author of this research:

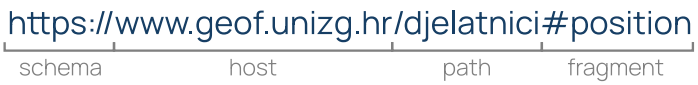


Figure 3.6
Example URI schema for web resource

As illustrated in Figure 3.6, URIs resemble URLs (Uniform Resource Locators) but differ in a key aspect that renders URLs less suitable for Semantic Web applications. URIs are intended to uniquely identify resources, whereas URLs are specifically designed to locate resources on the web. In this sense, URLs represent a specialized subset of URIs: all URLs are URIs, but not all URIs are URLs. A URI becomes a URL only when it explicitly specifies a way for accessing the resource on the web.

For the example in Table 3.1, we can identify all the resources in the triples by their URIs (Example 3.2).

Example 3.2

Triple no. 1:

(http://data.gov.hr/admUnits#Gvozd,

http://stat.gov.hr/census#hasPopulation,

"2047"^^http://www.w3.org/2001/XMLSchema#integer)

Triple no. 2:

(http://data.gov.hr/admUnits#Gvozd,

http://data.gov.hr/admUnits#withinCountry,

http://data.gov.hr/admUnits#CRO)

Writing URIs in their full syntax can be time-consuming; therefore, a shorthand notation, known as a Qualified Name (*qname*), is commonly used. A qname consists of a namespace prefix and a local name, separated by a colon. The namespace corresponds to the path required to reach the resource, while the local name identifies the specific resource within that namespace (Baučić, 2014). For the URI examples provided earlier (Example 3.2), the corresponding qnames can be represented as follows (Example 3.3):

Example 3.3

namespace

units: http://data.gov.hr/admUnits#

stat: http://stat.gov.hr/census#

xsd: http://www.w3.org/2001/XMLSchema#

Now, we can describe RDF triples like:

(units:Gvozd, stat:hasPopulation, "2047"^^xsd:integer), and

(units:Gvozd, units:withinCountry, units:CRO)

3.2.4 Literals and Blank nodes

As illustrated in Example 3.2, the object in Triple 1 is represented differently from the object in Triple 2. This distinction arises because RDF is fundamentally concerned with representing resources, which in some cases may be literal values, such as plain text, numbers, Booleans, or dates. The datatype of a resource is specified using a URI—for instance, an integer is represented as xsd:integer—while the actual value is included as a literal in the triple (e.g., 2047^^xsd:integer). It is important to note that literals are plain

values that cannot be further described; consequently, they can only appear as objects in RDF triples.

In some cases, a resource cannot be assigned a URI because its precise identity is unknown, yet certain information about it still needs to be represented in RDF. For example, consider Example 1: we know that *Gvozd* exists and has a population of 2047, but we do not know whether it represents a building, an administrative unit, or some other entity. Since no URI can be applied, RDF allows *Gvozd* to be represented as a blank node in triples, enabling the inclusion of additional information about it within the network. Blank nodes are distinguished from standard URI-based resources, often using a placeholder such as "?" to indicate that they do not correspond to a traditional URI.

3.2.5 Unicode

Resources identified by URIs may include specific characters, such as diacritics, or characters from region-specific alphabets, such as Cyrillic or Chinese. To represent these characters, computers employ various encoding systems, including ASCII, Latin-1, and Unicode. Originally, URIs were based on a subset of the ASCII standard, which supports only English-language characters (Hogan, 2020). Consequently, characters outside this subset must be encoded using substitution techniques, such as percent-encoding. For example, the blank space character is not part of the ASCII subset for URIs and is therefore encoded as *%20*; thus, the phrase admin units would be represented as *admin%20units*. To maintain readability for multiword resource names and avoid parsing or reasoning issues caused by encoding constraints, naming conventions such as CamelCase are often employed. In the CamelCase convention, each word begins with a capital letter, e.g. adminUnits, ensuring that URIs remain both human-readable and machine-processable.

To enhance character representation, the newer version of RDF (v1.1) introduces a more generalized identifier concept: the Internationalized Resource Identifier (IRI), which addresses the limitations of traditional URIs (Hogan, 2020). IRIs utilize the Unicode character encoding standard, which can represent over a million characters across all the world's languages (Unicode Consortium, 2024). For this reason, Unicode has become

universal encoding system (Al-Feel et al., 2008) with UTF-8 emerging as its most widely adopted encoding standard (W3Techs, 2025).

URIs and Unicode are well-established technologies originally developed for the traditional web. According to (Al-Feel et al., 2008), these technologies provide the essential capabilities of unique object identification and comprehensive character representation, which are required across the upper layers of the Semantic Web architecture. For this reason, Berners-Lee positioned them at the foundation of the Semantic Web technology stack.

3.2.6 RDF Serializations

RDF is a data model that represents data in RDF triples. To store, share, parse and process the data, RDF triples need to be serialized. Serialization means to write RDF triples in text with syntax that machines can process. Since the creation of Semantic Web, several syntaxes, more or less complex, have been proposed for these purposes.

RDF/XML

RDF/XML serialization is based on Extensible Markup Language (XML) whose hierarchical structure allows nesting of marked elements within the xml schema. XML is easily readable to both humans and computers and is used to represent structured data on the web so it can be stored, transmitted and reconstructed (W3C, 2008a). Its availability, simple structure and syntax that allows to markup data was the reason XML was recommended as a format to write down semantically enriched data in the early stages of Semanti Web (Hogan, 2020). Example 3.1 in RDF/XML syntax can be represented as (Example 3.4):

Example 3.4

```
<rdf:RDF
xmlns:units="http://data.gov.hr/admUnits#"
        <units:Gvozd
rdf:about="http://data.gov.hr/admUnits#Gvozd">
        <units:withinCountry>CRO</units:withinCountry>
        </units:Gvozd>
</rdf:RDF>
```

However, as XMLs can get very large and complex to read, and can represent the same information in myriad ways, RDF/XML is usually replaced by other, more simple syntaxes (Hogan, 2020).

N-Triples

The simplest syntax to represent RDF triples is N-Triples as it follows the logic in raw triples. It represents resources with their full URIs for every triple element within < > brackets and in a triple per line of text followed by ".". Even though simple, such representation is considered mayor drawback of N-triples as data can get complex to read. Example 1 represented in N-Triples serialization can be as follows Example 3.5):

Example 3.5

```
<http://data.gov.hr/admUnits#Gvozd> <http://stat.gov.hr/census#hasPopulation>
"2047"^^<http://www.w3.org/2001/XMLSchema#integer> .
```

Turtle

Turtle serialization combines triples representation of N-Triples with qnames. At the beginning, its states namespaces and represent triples line by line using local names. Example 3.1 written in Turtle serialization is as follows (Example 3.6):

Example 3.6

```
@prefix unit: http://data.gov.hr/admUnits#
@prefix stat: http://stat.gov.hr/census#
@prefix xsd: http://www.w3.org/2001/XMLSchema#
unit:Gvozd stat:hasPopulation 2024^^xsd:integer ;
        unit:withinCountry unit:CRO .
```

Among the revised three most common serializations, Turtle maintains the best easy human readability with machine-processibility all in one.

3.3 Schema and Ontology Languages

To provide structure and semantics to raw data, Semantic Web relies on schema and ontology languages. These languages, such as RDFS and OWL, define classes, properties, and relationships that describe how data is organized and how concepts interrelate. By

formalizing these patterns, they provide a framework for consistent interpretation and reasoning, ensuring that knowledge can be shared and understood across diverse systems.

3.3.1 RDF Schema

RDF addresses the problem of structural heterogeneity by unifying data from diverse sources within a single, flexible data model; however, it does not resolve semantic differences among the data. In other words, RDF provides a uniform graph structure but does not define the meaning of the resources within the graph or their interrelationships. The RDF Schema Language (RDFS) builds upon RDF, serving as a vocabulary that formally specifies the meaning and structure of RDF terms (W3C, 2014a). By introducing basic semantic descriptions for resources, RDFS establishes the foundational layer of semantics in the Semantic Web.

According to the RDFS specification, the RDFS language differentiates several types of resources, most notably classes and properties, which can be used to describe other resources, as well as the hierarchies and relationships among them (W3C, 2014a) (Table 3.2). Classes serve to categorize resources into groups, while properties establish connections between these groups. The W3C specification further defines a set of meta-classes within the rdfs namespace, providing lightweight semantics for structuring RDF data (W3C, 2014a):

Table 3.2
RDF Schema main classes (from W3C, 2014)

Name	Description
rdfs:Resource	Everything provided on the web is a resource. Therefore, everything on the web is a member of *rdfs:Resource* class.
rdfs:Class	Resources on the web can be grouped, and these groups are defined as members of *rdfs:Class* class.
rdfs:Literal	If resource on the web is literal value such as text or number, then it is member of *rdfs:Literal* class
rdfs:Datatype	If a resource describes data type (string for text, date, etc.) then it is a member of *rdfs:Datatype* class. It can only be object in RDF triple.

rdf:HTML	If a resource describes literal HTML values, then it is member of *rdfs:HTML* class.
rdf:XMLLiteral	If a resource describes literal XML values, then it is member of *rdfs:HTML* class.
rdf:Property	If a resource on the web is property – appearing as predicate in a triple, then it is member of *rdfs:Property* class.

Furthermore, the W3C specification defines a set of properties in RDFS that serve to establish connections between subject and object resources in RDF triples (Table 3.3). All of these properties are instances of the *rdf:Property* class and are constrained by their domain and range. Both the domain and range are also instances of *rdf:Property* and specify the types of resources that can appear as the subject (domain) or object (range) of a given property.

Table 3.3
RDF Schema selected properties (from W3C, 2014a)

Name	Description	Domain	Range
rdfs:domain	Domain definition. If P *rdfs:domain* C, subject in triples where P is predicate is instance of class C	rdfs:Property	rdfs:Class
rdfs:range	Range definition. In P *rdfs:range* C, object in triples where P is predicate is instance of class C	rdfs:Property	
rdf:type	Subject in the triple is instance of a class. In R *rdf:type* C, resource R is instance of class C	rdfs:Resource	
rdfs:subClassOf	Class is subclass of a class. In C1 *rdfs:subClassOf* C2, class C1 is subclass of class C2	rdfs:Class	
rdfs:subPropertyOf	Property is subproperty of a property. In P1 *rdfs:subPropertyOf* P2, property P1 is subproperty of P2	rdf:Property	rdf:Property
rdfs:label	Human-readable version of a resource's name.	rdfs:Resource	rdfs:Literal

	In R *rdfs:label* L, resource R has label L		
rdfs:comment	Human-readable description of resource. In R *rdfs:comment* L, resource L has comment L	rdfs:Resource	

To address more specific use cases, RDFS also includes additional classes and properties. Notably, it provides container and collection classes, along with associated properties, to describe and represent groups of resources. The key distinction between these two types is that collections are closed and represent a fixed set of terms, whereas containers can be expanded to include additional members. A representative example of a collection class is *rdf:List*, which is used to describe lists or list-like structures (W3C, 2014a).

Although relatively simple, the RDFS schema provides a powerful vocabulary that effectively addresses challenges in integrating data from heterogeneous sources. In RDF graphs, both the data and its schema are stored together, offering flexibility and agility in data manipulation, features that are particularly valuable for data integration. According to Baučić (2014), RDFS provides three main benefits related to data integration:

- Terminology reconciliation in which *rdfs:subClassOf* and *rdfs:subPropertyOf* allows to reconcile terms by establishing their hierarchy. This way, no new semantics are needed and terminology from existing sources can still be used in the same way. Such approach is highly appreciated by computer systems and applications that do not need to be altered in case of data new data integrated in the graph.
- Merging similar data coming from different source can be easily done with the introduction of new classes. These new classes serve as a bridge between data semantics and again allow linking of data from different sources without altering ways data is used.
- Filtering and classification of data based on their domain and range definitions allows new classifications of data into new classes of the same type which don't affect how data used outside of the web.

3.3.2 Web Ontology Language

While RDFS provides basic semantics that enable reasoning about data structure and relationships, it lacks the expressive power required to represent more complex real-world scenarios, such as specifying that two classes are disjoint. The Web Ontology Language (OWL) addresses this limitation by extending the RDFS vocabulary with additional terms that support richer semantic expression (Hogan, 2020). OWL introduces an expanded set of constructs for formally defining classes, properties, individuals, and the logical relationships among them. These constructs include, among others, cardinality constraints to express multiplicity, property characteristics, and enumerated classes.

Within OWL, three sublanguages exist that differ in their semantic expressivity: OWL Lite, OWL DL, and OWL Full. OWL Lite is the simplest of the three and is primarily used for defining classification hierarchies and simple constraints, such as cardinality. Its constructs are shared with OWL DL and OWL Full, but differ in how they are restricted for use. OWL DL is a highly expressive language based on description logic, enabling structured knowledge representation while remaining computationally feasible. This computational feasibility is maintained through restrictions on OWL Lite constructs; for example, a class may be a subclass of another class, but it cannot simultaneously be an instance of that class. Both OWL Lite and OWL DL impose constraints on construct usage, effectively functioning as restricted forms of RDF. In contrast, OWL Full is considered a true extension of RDF, imposing no restrictions on constructs, anything deemed reasonable is permitted. However, this unrestricted nature renders computation potentially undecidable, meaning results are not guaranteed (W3C, 2004b). Some of OWL constructs defined by W3C are summarized in Table 3.4:

Table 3.4
Selected constructs from Web Ontology Language (from W3C, 2004b)

Constructor	Description
PROPERTIES	
owl:ObjectProperty	If predicate P is of type ObjectProperty, it links two instances.
owl:DatatypeProperty	If predicate P is of type DatatypeProperty, it links instance to datatype.

owl:AnnotationProperty	Attaches metadata such as comment or label to classes, properties, individuals and ontology headers.
OBJECT PROPERTIES CHARACTERISTICS	
owl:FunctionalProperty	Subject instance can be linked to only one object instance. If *p* is functional property and C1 *p* C2 and C1 *p* C3, then C2 and C3 are the same instances.
owl:InverseFunctionalProperty	Object instance can be linked to only one subject instance. If *p* is inverse functional property and C1 *p* C2 and C3 *p* C2, then C1 and C3 are the same instances.
owl:TransitiveProperty	If *p* is transitive property and C1 *p* C2 and C2 *p* C3, then C1 *p* C3.
owl:SymmetricProperty	If *p* is symmetric property and C1 *p* C2, then C2 *p* C1.
RESTRICTIONS	
owl:Restriction	Class that represents a set of individuals satisfying a specific constraint on a given property.
owl:onProperty	Attaches restriction to a property.
owl:hasValue	Defines a class of all individuals for which a given property has a specific, required value.
owl:someValuesFrom	Defines a class of all individuals for which at least one value of a given property belongs to a specified class.
owl:allValuesFrom	Defines a class of all individuals for which every value of a given property (if any exist) must belong to a specified class.
EQUALITY	
owl:equivalentClass	Asserts that two classes have the exact same instances.
owl:equivalentProperty	Asserts that two properties relate the same pairs of individuals.
owl:sameAs	Asserts that two URIs refer to the exact same individual.
CLASS OPERATIONS	
owl:intersectionOf	Defines a class as the intersection of two or more other classes.
owl:unionOf	Defines a class as the union of two or more other classes.
CARDINALITY	
owl:cardinality	Restricts a property to have exactly N distinct values for an individual.
owl:minCardinality	Restricts a property to have minimum of N distinct values for an individual.
owl:maxCardinality	Restricts a property to have maximum of N distinct values for an individual.

OWL constructors in the Semantic Web are grounded in SPARQL CONSTRUCT queries. By specifying patterns in the WHERE clause of a query, new semantic information can be generated and added to the RDF graph, thereby enriching the existing knowledge base.

The constructors described in Table 3.4 enable the addition of more specific semantics to the RDF graph. Depending on user requirements, these constructors can be combined to create more complex and expressive statements, for example, defining a class as the union of intersections, where instances must satisfy particular type constraints. In OWL, such restrictions are especially important, as they allow data to be maintained in their natural form while enabling reasoning based on constraints imposed on properties. In certain cases, RDF triples may conflict with OWL-defined logic, resulting in a logically inconsistent model (Baučić, 2014).

3.4 Knowledge Representation and Reasoning

Building on structured schemas, knowledge representation formalizes information in a way that machines can interpret and manipulate. Description logic provides logical foundation for this process, while reasoners leverage this structure to infer implicit knowledge, check consistency, and ensure that the knowledge base aligns with its logical constraints.

3.4.1 Description Logic

The central idea of the Semantic Web is that machines should be able to read, interpret, and automatically reason over data. For instance, if it is defined that "Fido is a dog" and "All dogs are animals," a machine should be able to infer that Fido is an animal without this being explicitly stated. Achieving such reasoning requires rigorous mathematical foundations to bridge the gap between human conceptualizations of the world and the computational mechanisms necessary for intelligent systems to process, query, and derive new knowledge. These foundations are provided by Description Logic.

Description Logic is a fragment of First-order logic, a standard for the formalization of mathematics into axioms, optimised specifically for knowledge representation in ontologies. Its expressivity is restricted to (only) logical axioms of concepts (classes in ontology), roles (properties in ontology), individuals and their combination to represent

knowledge. Concepts in the Description logic correspond to unary predicate symbols in First-order logic (Class(x)), roles to binary predicates (predicate(x,y)) and instances to constants (Constant). Because of this restricted expressivity, Description Logic is decidable, i.e. guarantees reliable, terminate and efficient automated reasoning in contrast to First-order logic which suffers from undecidability, meaning that some reasoning tasks may not terminate or require infinite computation (Hogan, 2020).

Formally speaking, Description Logic is a family of formal knowledge representation languages based on a combination of attributive language ($\mathcal{AL}$) and complement ($\mathcal{C}$). While $\mathcal{AL}$ defined classes by stating attributes their members must have (that is where the name comes from), it lacked negation expressivity. Negation in $\mathcal{AL}$ was introduced by the Complement $\mathcal{C}$ and made $\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplement ($\mathcal{ALC}$) a foundation language of the Description Logic.

$\mathcal{ALC}$ language is founded on atomic concepts and atomic roles which are combined to construct more complex concepts and roles, and individuals are treated as instances of concepts, always atomic (Krisnadhi & Hitzler, 2014). Hogan (2020) provides description of $\mathcal{ALC}$ constructs and links them to corresponding terms in OWL DL language (Table 3.5).

Table 3.5
ALC language concepts with corresponding description logic semantic and owl equivalent (summarised from Hogan, 2020)

Name	Syntax	Semantics	OWL key-term
Atomic concept	A	$A^I \sqsubseteq \Delta^I$	owl:Class
Atomic property	R	$R^I \sqsubseteq \Delta^I \times \Delta^I$	owl:Property
Individual	a	$a^I \in \Delta^I$	RDF URI or Literal
Top concept	$\top$	Δ^I	owl:Thing
Bottom concept	$\bot$	$\varnothing$	owl:Nothing

Concept negation	¬ C	$\Delta^I \setminus C^I$	owl:complementOf
Concept intersection	C ⊓ D	$C^I \cap D^I$	owl:intersectionOf
Concept union	C ⊔ D	$C^I \cup D^I$	owl:unionOf
Existential restriction	∃R.C	$\{x \mid \exists y:(x,y) \in R^I \text{ and } y \in C^I$	owl:someValuesFrom
Universal restriction	∀ R.C	$\{x \mid \forall y:(x,y) \in R^I \text{ implies } y \in C^I$	owl:allValuesFrom

According to Hogan (2020) interpretation of these constructs *I* in Description Logic is described by a pair $(\Delta^I, \cdot^I)$ where Δ^I is the nonempty domain and $\cdot^I$ is the interpretation function that maps every individual, concept and role to their respective elements in the domain Δ^I. For the constructs in the table (Table 3.5) this means the following. Atomic concept A is mapped to A^I which is an element of the domain Δ^I, and atomic role R is mapped to binary relation R^I, defined as subset of Δ^I x Δ^I. Individual a is mapped to a^I, an element of Δ^I and ⊤ and ⊥ are concepts that contain all or none of the individuals in the domain. Negation concept ¬ C expresses the set of all individuals that do not belong to the interpretation of concept C (complement to), C ⊓ D and C ⊔ D are intersection and union of concepts, and ∃R.C and ∀ R.C are restrictions on properties that define individuals as instances of C.

These main constructs of $\mathcal{ALC}$ can be extended with new constructs to allow higher expressivity of Description Logic. For example, $\mathcal{S}$ adds transitive closure, $\mathcal{H}$ adds inclusion role, $\mathcal{I}$ adds inverse roles. In such a way, new languages with more expressivities can be built (Hogan, 2020):

$$[\mathcal{ALC} \mid \mathcal{S}][\mathcal{H} \mid \mathcal{R}][\mathcal{O}][\mathcal{I}][\mathcal{F} \mid \mathcal{N} \mid \mathcal{Q}]$$

In OWL 2 DL, most recent version of OWL, semantic expressivity is underlined by $\mathcal{SROIQ}$ ($\mathcal{D}$) Descriptive Logic language which means it supports transitivity ($\mathcal{S}$), role inclusion ($\mathcal{R}$), nominals (enumeration lists), inverse roles ($\mathcal{O}$), inverse roles ($\mathcal{I}$), qualified number restrictions ($\mathcal{Q}$) and different data types ($\mathcal{D}$) (Hogan, 2020).

3.4.2 Knowledge representation in Description Logic

Knowledge in ontologies is represented in axioms composed of concepts, roles and individuals and is stored in Description Logic Knowledge Base (Hogan, 2020). This Knowledge Base (K) can formally be described as triple K(T, R, A), where T stands for Terminological Box (TBox), R for Role Box (RBox) and A for Assertion Box (ABox) (Krisnadhi & Hitzler, 2014).

TBox and RBox specify knowledge of a domain by defining a vocabulary of concepts (classes) and roles (properties), together with axioms that capture hierarchical relationships, constraints, and logical dependencies between them (Borgida & Brachman, 2010). For example, if class C is subclass of class D (C⊑D), TBox will contain axiom $C^{I} \subseteq D^{I}$ which has the meaning of rdfs:subClassOf in RDF Schema. Same holds for properties, but these axioms are stored in RBox. In literature (Giacomo & Lenzerini, 1996; Nardi & Brachman, 2010) TBox is used as a single term for both TBox and RBox as they both describe schema of the domain, but growing expressivity of role axioms urges for separation. This is notable in OWL 2 DL which is based on $\mathcal{SROIQ}$ where rules for ensuring decidability, such as role of hierarchy, are centered on analysing properties (RBox).

While TBox and RBox axioms specify intensional knowledge, ABox axioms encode extensional knowledge consisting of assertions about specific individuals; their membership in concepts (a: C) and participation in roles, (a,b) : R (Krisnadhi & Hitzler, 2014). For example, if Sava is an instance of river, and it flows through Zagreb, an instance of city, ABox assertions will have axioms (Example 3.7):

Example 3.7

Sava: river, Zagreb: city, (Sava, Zagreb) : flows through.

Axioms in the TBox, RBox, and ABox of an ontology provide a formalized representation of domain knowledge, serving as the foundation for reasoning procedures that derive logical consequences. Reasoning over TBox and RBox axioms enables the classification of concepts, the discovery of implicit subsumption relationships, and the inference of constraints and hierarchies among properties, including role inclusions, transitivity, and inverses. In the ABox, reasoning facilitates tasks such as consistency checking, instance

classification, and query answering by verifying whether individual assertions conform to the constraints defined by the schema axioms. Overall, reasoners ensure that the knowledge base remains logically coherent while also enriching it with new knowledge derived from explicitly asserted axioms.

3.5 SPARQL

Data in Semantic Web are represented in RDF triples which form directed graph of Semantic Web. To access and manipulates data in such a structure, appropriate query and update languages should be used.

SPARQL (SPARQL Protocol And RDF Query Language) is a standard query language proposed by the W3C (W3C, 2008b) to access data in the graph-based structure of Semantic Web. As a query language, it shares common characteristics with other query languages (Allemang & Hendler, 2011). For example, Baučić (2014) describes that the structure of SPARQL query is similar to structure of SQL in terms of meaning of used key words, e.g. Select, Distinct, Filter, Order by, Limit. However, the main difference between them is that SPARQL does not cross-reference data as these references are already contained within the RDF data model (Baučić, 2014).

The simplicity of SPARQL query language is in the foundational notion of reusing triple pattern from RDF model so queries look like data statements but with a question word in a position of triple's unit of interest (Allemang & Hendler, 2011). This allows powerful queries across the graph as subjects and objects (when not literals) can take part in multiple triples. Similarly to other query languages, SPARQL operations can be divided between reading data from the graph (query) and modifying the graph (update).

3.5.1 SPARQL Query

To extract data from RDF triples described in Example 2, a simple SPARQL query can be given as (Example 3.8):

Example 3.8

```
SELECT ?unit
WHERE {
        admUnit:Gvozd admUnit:withinCountry ?unit .
}
```

In the example above, SPARQL SELECT query was used to extract information about which country municipality of Gvozd is located within. To get this data, the query contains two main parts, SELECT and WHERE. In the SELECT part, one or more arbitrary question words (with ? prefix) are used to indicate data to be extracted from the graph. In the WHERE part, also called graph pattern (Allemang & Hendler, 2011), these question words are reused in the correct position within a triple. Statements in the WHERE part are then compared with statements in data graph, and once the exact match is found, resource behind question word is returned as a result. For the example above, the return value for the query would be *admUnit:CRO URI*.

Queries can be more complex than this. In that case, the WHERE part contains more statements that should be compared to data graph to get the result. For example, if another RDF triple (admUnit:CRO, admUnit:inContinent, admUnit:Europe) is added to the graph, following the same node URI, admUnit:CRO object from the previous example becomes subject in the new triple. If query is to extract continent municipality of Gvozd is in, the query would have the following structure (Example 3.9):

Example 3.9

```
SELECT ?continent
WHERE {
        admUnit:Gvozd admUnit:withinCountry ?unit .
        ?unit admUnit:inContinent ?continent .
}
```

In this example, the query engine will take the first WHERE statement, look up for *?unit* in the data graph and extract URI from the matching triple. The URI will then be used in the second statement to look for the triple with *admUnit:incontinent* predicate and return *admUnit:Europe* result.

When having more than one statement in the query, the order of the statements is not important (Allemang & Hendler, 2011). This is because the semantics are contained within the RDF model, and for the result to be found, all statements must be fulfilled either way. Ordering of statements, however, makes sense for the faster query execution. If statements are ordered in a way such that every new statement narrows down the number

of triples to look up for, the query will be executed faster. For this reason, single-question word statements are put first in the WHERE part of the query.

So far, querying was illustrated on examples where only data on subjects and objects in RDF triples was extracted from the data graph, but querying can also be used to search for the relation properties between them. In this case, question word is placed in the position of the predicate in the query statement. However, if subject of the triple uses the same predicate to point to different objects, such a query will return multiple predicates (properties) of the same type. For example, if the municipality of Gvozd is attached another population data literal using the same stat:hasPopulation predicate, e.g. from penultimate census (not concerning time reference), querying the graph for predicate would return two same values. To filter out only representative types, SPARQL includes DISTINCT keyword in the SELECT part of the query (Example 3.10) (Allemang & Hendler, 2011):

Example 3.10

```
SELECT DISTINCT ?predicate
WHERE {
        admUnit:Gvozd ?predicate ?value .
}
```

Similarly to removing duplicate predicates, SPARQL offers FILTER keyword to make more precise statements about the data to be extracted from the graph. For example, if the query is to look for population data of municipality of Gvozd that is lower than 2000 people, the query would be structured as follows (Example 3.11):

Example 3.11

```
SELECT ?population
WHERE {
        admUnit:Gvozd stat:hasPopulation ?population .
        FILTER (?population > 2000^^xsd:integer)
}
```

Apart from SELECT queries which return a table of values, SPARQL supports ASK queries with Boolean return type (Yes and No), CONSTRUCT queries returning an RDF graph, and DESCRIBE query that returns RDF graph describing terms or solutions

(Hogan, 2020). The resulting RDF graphs from CONSTRUCT can be returned to the source graph or can be extracted into a new graph, depending on the user needs.

Query functionalities of SPARQL support extracting data from data graph, but cannot meet the requirements of inserting new, external data. For these purposes, SPARQL Update language should be used.

3.5.2 SPARQL Update

As defined by W3C, SPARQL update is a companion language to SPARQL query and is used for executing updates to the RDF data graph (W3C, 2013). It reuses SPARQL query syntax and allows to insert, delete, load and clear data graph as well as to create new and drop existing graphs.

Instead of writing new triples when data is to be added to the graph, SPARQL update allows to add new content using query pattern (Hogan, 2020). Here, instead of SELECT and WHERE, it uses INSERT DATA statements to express what the query should insert. For example, if area value is the new information to be attached to admUnit:Gvozd resource in the graph, the insert triple query will have the following form (Example 3.12):

Example 3.12

```
namespace
geo: http://example.com/geometry/
xsd: http://www.w3.org/2001/XMLSchema#
INSERT DATA {
        admUnit:Gvozd geo:hasArea "212.4"^^xsd:double .
}
```

Although primarily designed as a query language for semantic web, SPARQL is also a communication protocol ("P" in the SPARQL) that allows communication of client with the server in the context of web services (Allemang & Hendler, 2011). In SPARQL, server for this protocol is called SPARQL Endpoint and represents the access point to knowledge stored in data graph (triplestore). When trying to access the data, client sends a request containing SPARQL query to Endpoint's URL, Endpoint executes this request against the data in the graph and sends query results back to the client in the requested format (e.g.

JSON). Setting up a SPARQL Endpoint can rely on predefined solutions such as Docker platform (Docker, 2025) or GraphDB SPARQL Endpoint (Ontotext, 2025).

3.6 Semantic Web and Population Disaggregation

Spatial data integration is the central part of population disaggregation and Hasani et al. (2015) emphasize that the semantic web has the potential to improve this integration through exploitation of ontologies and RDF data model. Further on, Muscetti et al. (2022) suggest that managing the increasing amount of available data requires automated processes, which De Meester et al. (2020) associate with the benefits of Semantic Web. This can be directly applied to the domain of population disaggregation where geospatial data is constantly included in newly developed methods seeking improved disaggregation accuracy. This means Semantic Web applications are becoming crucial for automated executions (De Souza Neto et al., 2018; Bednár et al., 2024). However, despite their potential, functionalities of the semantic web in population data domain are currently being used mainly to disseminate semantically enriched data (e.g., Wong et al., 2024).

Literature review shows limited availability of ontologies describing population disaggregation as a process which clearly indicates that research in this area is highly needed, especially given the relevance of population data in modern day decision making. To best of authors knowledge, the only approach to semantical modelling of population disaggregation is proposed by King (2019). In her work, King proposes a semantical framework for very detailed population counts estimate, at address level, in a continuous temporal scale that observes how population changes in time and space. To achieve this, the framework is divided into three parts modelling spatial, temporal and attribute domains. Spatial domain details how population is allocated in space, temporal domain monitors how population changes over continuous time and attribute domain attaches attributes such as activity types that might affect population presence or absence in space and time. To estimate population with high spatial resolution and in continuous time, the ontology uses geographic data (addresses, topographic objects) in combination with population-informative data (residence, visitor data, ...). To model the data, ontology introduces three high-level classes: Region, Place and Temporal Signature, and subclasses it for specific needs of data. For example, Residential buildings are subtype of Address

which is a subtype of Place. Place is the main concept of the ontology and represents locations where people engage in some kind of activity, residential, working or leisure. Places are linked to Regions, e.g. statistical regions or topographic areas, and to Temporal Signature that define its temporal span (King, 2019). By setting the ontology this way, the model is capable to observe spatial distribution of population across very detailed time references and reason about population counts on a specific location in a given moment of time.

Even though highly detailed, such approach features disadvantages within a broader application context. The most limiting factor of the model is that it relies on availability of high-resolution spatiotemporal population data to estimate population counts. This causes model not to be universally applicable as such data are not always available to users. Further on, framework uses one disaggregation method which is firmly incorporated into ontology structure. This is beneficial when such detailed results are needed, but in some cases when users have limited access to data and only need informative population counts, e.g. in preliminary analysis for urban planning, other disaggregation methods could better serve the needs.

As described in section 2.2.5 (Practical Implementation of Spatial Population Disaggregation), scientific community emphasizes that disaggregation domain lacks practical solutions that would support widespread creation of high detailed population data. While model proposed by (King, 2019) contributes to it from a scientific point of view, it does not consider existing approaches to disaggregation methods. A solution that would combine both, a population disaggregation as a process, and potential of semantic web automation would therefore better support needs of the community.

According to De Meester et al. (2016), central to automation are semantically described executable procedures that define tasks, such as algorithms and parameters, and their connection to results. By modelling procedures as concepts within an ontology, the model becomes flexible, enabling the inclusion of new methods, automatic procedure discovery and execution, and reuse of the model because the conceptual modelling is independent of data instances.

Benefits of modelling population disaggregation as a process within an ontology are multiple. Ontology represents formal knowledge about the domain which means it is capable of capturing domain's main concepts and their mutual relations. This will allow expansion of semantic web knowledge base and allow reuse of this specific knowledge in other alike applications. Furthermore, ontologies are based on RDF, a simple data model that effectively bridges the issue of integration of data from different sources. This means that published data will be easily integrated in the disaggregation process while publication of new knowledge and data could seamlessly lead to more accurate population estimations, which is another important benefit.

Creation of spatiotemporal population disaggregation ontology is based on the understanding of its basic requirements. As explained in the section 2.2.1, population data tied to a source zone is being disaggregated to target zones using source zone's characteristics or spatial ancillary data. This implies that the disaggregation process relies on integration of statistical data (population counts) with geospatial data (source zone, ancillary data) while considering their time references. From semantical perspective, disaggregation ontology should therefore model a process that integrates data from three different domain ontologies. In line with the best recommendations for semantic web development, where reuse of existing knowledge should be prioritised rather that creating new knowledge, the following chapter will make a review of existing ontologies relevant for population disaggregation process.

3.7 Research Relevant Domain and Task Ontologies

Within the Semantic Web, numerous ontologies have been developed to describe geospatial, statistical, temporal, and process-related domains. While these ontologies often refer to the same underlying concepts, they represent them differently to accommodate specific user requirements and perspectives.

3.7.1 Geospatial Domain Ontologies

According to Kaladzavi et al. (2017), geospatial data is described in three data levels: semantic, geometric and spatial relations level. Semantic level deals with the nature and aspect of objects, spatial relations observe how object relate and interact with other objects while geometric level deals with representation of their shape and location on the surface

of the Earth. Based on this classification, different ontologies for different data levels can be developed.

Collection of ontologies SWEET conceptualizes knowledge of space for Earth related sciences using OWL language. Developed in 2004, SWEET act as an upper-level ontologies for geospatial data in which spatial extents, e.g. country, and relations, e.g. aboveOf, are considered special cases of numeric extents and relations. Due to its broad meaning concepts, this ontology is not suitable for detailed domain knowledge representation and is rather use for interoperability among systems (Raskin & Pan, 2005).

The GeoNames ontology is a formal representation of geospatial knowledge providing a shared vocabulary for describing places and their relationships, serving as schema layer for GeoNames database (Tao et al., 2024). All features in the ontology are represented as points and described using SKOS ontology (Simple Concept Organization System) while their features are described in OWL. The ontology is used to describe concepts such as names, places, administrative subdivisions, and uses latitude and longitude coordinates of World Geodetic System 1984 (WGS84) to represent position of the feature (Kaladzavi et al., 2017). While it does support point feature representation, GeoNames does not support other geometric representation types, such as lines and polygons which limits its usability in applications interested in geometric relations among features, e.g. population disaggregation methods (GeoNames, 2025).

The iCity-Geometry Ontology is a vocabulary designed for representing and managing geometric information within the context of urban environments and smart cities. It builds upon existing geometry concepts like points, lines and polygons, but tailors them for the specific needs of urban informatics and intelligent city applications. The ontology includes data property asWKT to allow geometry representation in Well Known Text and offers limited types of spatial relation, e.g. exterior, interior, partOf (Katsumi, 2025).

Today, the most common vocabulary in the domain of geospatial data is GeoSPARQL, a standard proposed by the Open Geospatial Consortium (OGC). Introduced for representation of geospatial data in RDF, GeoSPARQL allows not only to represent the

data but to spatially reason about it which is highly beneficial when moving from data representation to data exploitation (OGC, 2012).

In GeoSPARQL, geospatial data is represented using three main classes: Spatial Object, Feature and Geometry. While Spatial Object describes anything that exists in space, strictly defined as to have shape or position, Features represent discreet object that can be uniquely identified. Geometry of such features is represented separately, using Geometry class. This dual representation of a single object using individuals of Feature and Geometry allows to make independent statements about what features are and how they are represented which at last makes it possible to reason separately over each aspect of the object. To link feature with its geometric representation, hasGeometry property is used (Figure 3.7) (OGC, 2024).

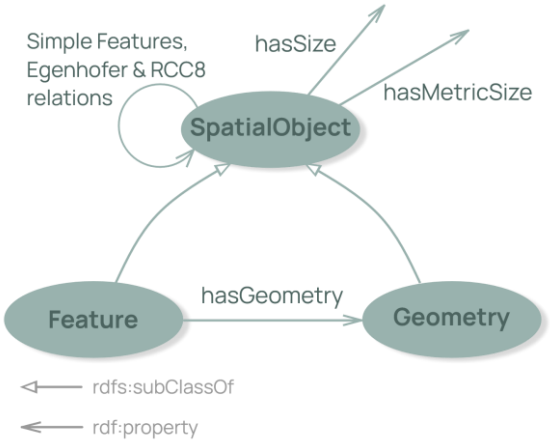


Figure 3.7
GeoSPARQL vocabulary core classes and properties. Simple Features, Egenhofer & RCC8 represent groups of qualitative spatial relations (from OGC, 2024)

To fully describe geospatial domain, three main classes have standard properties that are used to make statements about represented objects. These properties are not mandatory, but optional depending on the use case. For example, Spatial Object class uses hasMetricSize property to indicate the length of the object in meters and Feature uses hasCentroid property to link its individual to point representing centroid of its geometry. For Geometry class, formal properties define how geometry must be structured. For instance, hasSerialization links geometry individual with its text serialization while asWkt, a subproperty of hasSerialization, describes that the geometry format must be of type

Well Known Text. GeoSPARQL's inclusion of WKT geometry representation makes a direct connection to definition and hierarchy of geometries defined in OGC Simple Feature Standard. This not only ensures interoperability but also impacts way objects are perceived and conceptualized. More comprehensive list of properties tied to GeoSPARQL main classes is given in table (Table 3.6):

Table 3.6
GeoSPARQL vocabulary main classes and properties (summarised from OGC, 2024)

Name	Definition	Domain	Range
geo:SpatialObject			
geo:hasMetricSize geo:hasMetricLength geo:hasMetricArea geo:hasMetricVolume	The size/ length/ area/ volume of Spatial Object in meters	geo:SpatialObject	xsd:double*
geo:Feature			
geo:hasGeometry	Spatial representation for a given feature	geo:Feature	geo:Geometry
geo:hasBoundingBox	Minimal enclosing box of a feature		
geo:hasCentroid	Aritmetic mean position of all geometry points		
geo:Geometry			
geo:coordinateDimension	Number of axes in definition of CRS	geo:Geometry	xsd:integer
geo:isEmpty	If geometry has no information		xsd:boolean
geo:hasSerialization	Links geometry with text-based serialization		rdfs:Literal
geo:asWKT	WKT/ GML/ geoJSON serialization of geometry		geo:wktLiteral
geo:asGML			geo:gmlLiteral
geo:asGeoJSON			geo:geoJSONLiteral

*xsd: http://www.w3.org/2001/XMLSchema#

Unlike other geospatial ontologies, GeoSPARQL includes qualitative and quantitative spatial constructors that allow it to perform spatial reasoning. With these constructors, the machine is able to reason about spatial relations between objects e.g. proximity of two spatial objects (metric distance). GeoSPARQL provides three main sets of topological relation predicates (qualitative constructs), Simple Feature, RCC8 and Egenhofer, giving users a flexibility based on their needs and the underlying data's level of formal rigor. With these relations, computer can infer which geometries are equal, disjoint, contained or within other geometries or intersect, touch, cross and overlap (OGC, 2024).

The use of GeoSPARQL vocabulary in population disaggregation ontology can be beneficial for two main reasons. Firstly, wide adoption of the vocabulary ensures diversity of geospatial data that could be used to improve accuracy of disaggregation results. Secondly, its spatial reasoning capabilities tend to meet the requirements of spatial analysis which disaggregation is founded on. These make GeoSPARQL highly applicable in the domain of population disaggregation modelling.

3.7.2 Statistical Domain Ontologies

In the domain of statistical data, several ontologies exist. These ontologies describe different aspects of statistical data, ranging from production to representation. For example, STATO (Statistics Ontology) is oriented towards statistical data creation process and includes descriptions of statistical tests, their conditions of application and resulting outcomes (STATO Project, 2014) rather than statistical data itself. Following, GSIM ontology (General Statistics Information Model) is a complex, top-level domain ontology used to define, manage and use data and metadata in statistical production process. It describes classes which are input and output in production of statistics and model concepts in four major sections, business, exchange, concepts and structures (Dreyer et al., 2016).

The SDMX (Statistical Data and Metadata eXchange) Ontology is a formal representation of the SDMX standard by ISO (ISO:TS 17369), which is a foundational framework for the exchange of statistical data and metadata. The SDMX ontology structures statistical data as observations organized by dimensions, measures, and attributes, grounded in code lists and concept schemes, and tied together in datasets by

data structure definitions (Cyganiak et al., 2010). As SDMX is heavy and detailed, its use is more tied to institutional internal data exchange where data must strictly conform to SDMX standard. In contrary, for external dissemination of data, a W3C recommendation RDF Data Cube (QB) vocabulary is more convenient. QB is inspired by the SDMX ontology, shares the same concepts, but is lighter and prioritise ease of use. Main QB components are represented in Figure 3.8.

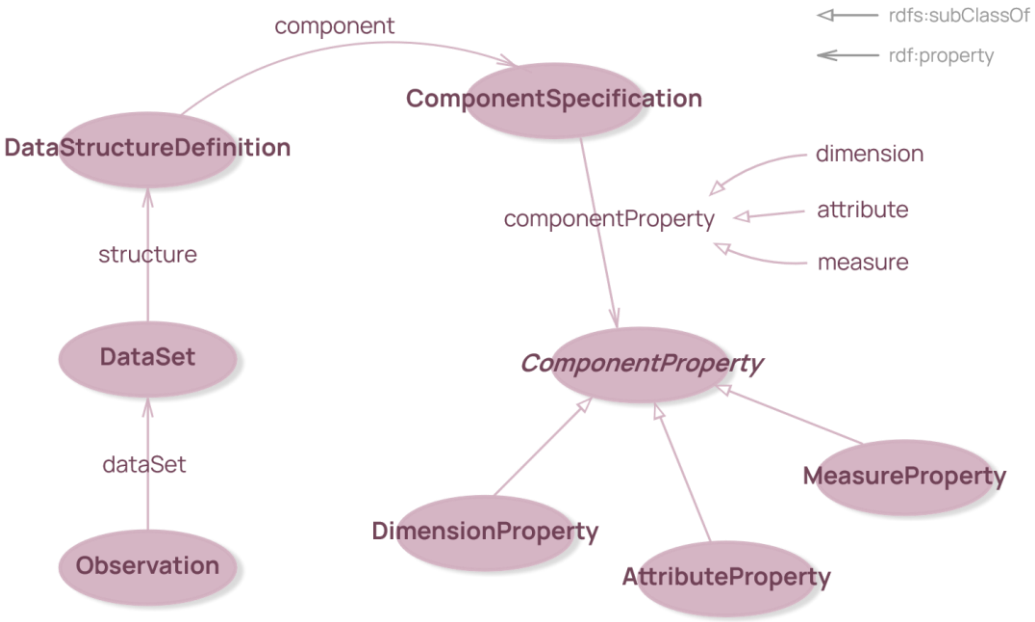


Figure 3.8
RDD Data Cube vocabulary main classes and properties (modified from W3C, 2014b)

QB uses several main classes, *qb:DataStructureDefinition*, *qb:ComponentProperty*, *qb:DataSet* and *qb:Observation* to describe statistical data (Figure 3.8). Data Structure Definition class sets structure for organization of statistical data according to dimensions, measures and attributes, collectively referred to as components. Dimensions describe what the observation applies to, e.g. spatial unit, measures name the phenomena observed and attributes add metadata, e.g. unit of measurement (W3C, 2014b). Framework defined by Data Structure Definition is used to create dataset which stores all individuals of observations (real data). To connect all the classes, the ontology uses several main properties: *qb:structure* to connect *qb:DataSet* with *qb:DataStructureDefinition*, *qb:component* to connect *qb:DataStructureDefinition* with *qb:ComponentProperty*, and *qb:dataSet* to connect *qb:Observation* with *qb:DataSet*. Flexibility and wide adoption of

QB vocabulary in representation of statistical data makes it applicable for modelling of population in population disaggregation process.

3.7.3 Time Domain Ontologies

Modelling time in ontology can be general-purpose or tied to a specific use case. For example, Time event ontology (TEO) is developed specifically to model clinical applications (Li et al., 2020). On the other side, Ontology of Time for the Semantic Web provides a structured framework for representing temporal knowledge, distinguishing between instants (points) and intervals (spans), and defining their relationships through primitives like before, begins, and inside. The ontology models durations, calendar and clock units, and time zones, enabling annotation of events with both abstract measures of time and concrete calendar references. While designed to support reasoning over temporal constraints on the Semantic Web, it suffers from computational complexity and limited compatibility with OWL reasoning (Hemalatha et al., 2012; Hobbs & Pan, 2004). OWL Time, a W3C candidate recommendation, is a lightweight ontology for modelling temporal concepts in Semantic Web. It models time through *time:Instant* and *time:Interval* classes, subclasses of a top-level class *time:TemporalEntity* (Figure 3.9). Instant class refers to fixed points in time while Interval describes spans from start instant to end instant. It also provides properties to express relations like ordering (*time:before*, *time:after*), containment (*time:inside*) and relative positions in time (*time:hasBeginning*, *time:hasEnd*) (W3C, 2022). Simplicity of time semantic descriptions makes OWL-Time easily applicable for the modelling of temporal component in proposed population disaggregation ontology.

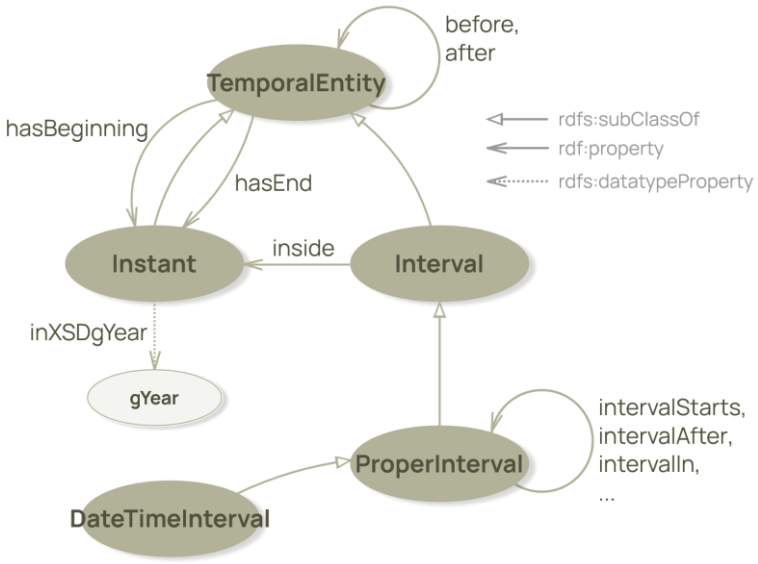


Figure 3.9
OWL-Time core classes and properties (modified from W3C, 2022)

3.7.4 Task ontologies

Modelling tasks as part of problem-solving activities can rely on different ontologies. For example, Semantic Sensor Network (SSN) ontology, based on Sensor, Observation, Sample and Actuator (SOSA) ontology, is comprehensive ontology for describing sensors, their observations, the systems they are part of, and the related processes. Its Procedure module contains three main classes, *sosa:Procedure*, *ssn:Intput* and *ssn:Output* that are used to describe a workflow, plan or computational method, i.e. steps that lead to reproducible results (Figure 3.10). Procedure class relates to *ssn:Input* and *ssn:Output* via *ssn:hasInputOnly* and *ssn:hasOutputOnly* properties. Classes of Procedure module are linked to classes of other modules to comprehensively model the entire process, from system description, features to results (W3C, 2017).

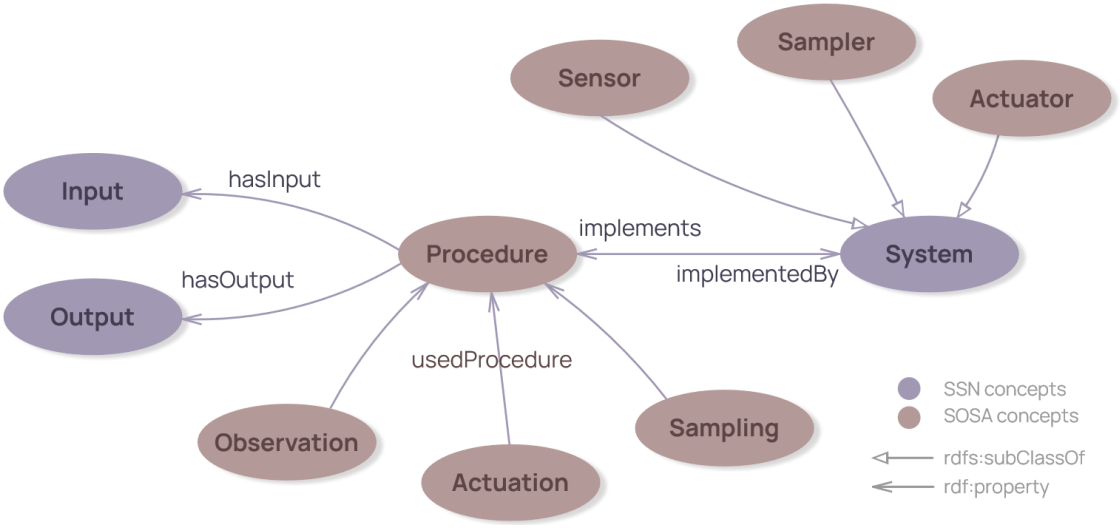


Figure 3.10
Semantic Sensor Network ontology procedure related classes and relations (modified from W3C, 2017)

Ontology Web Language for Services (OWL-S) is an upper-level ontology designed to describe properties and capabilities of web services in computer-interpretable form that enables automatic discovery, invocation, composition and monitoring of web services. The ontology consists of three main components, Service profile, Service grounding and Service model used to describe what the service does, how to access it and how it does it. The Service profile covers information transformation functionality of the service that deals with representation of inputs and outputs. Main class of this profile is Profile which uses *hasParameter* and its subproperties *hasInput* and *hasOutput* to describe what service model expects as input and output. Within the Service Model, the Process Ontology defines classes such as *AtomicProcess*, *CompositeProcess*, and *SimpleProcess*, and uses properties like *hasInput*, *hasOutput*, *hasPrecondition*, and *hasEffect* to formally describe the behavior of services (Figure 3.11) (W3C, 2004a).

Even though OWL-S ontology describes problem-solving methods, tasks and activities in domain independent way and is highly expressive, it expressiveness can make it too complex for efficient reasoning.

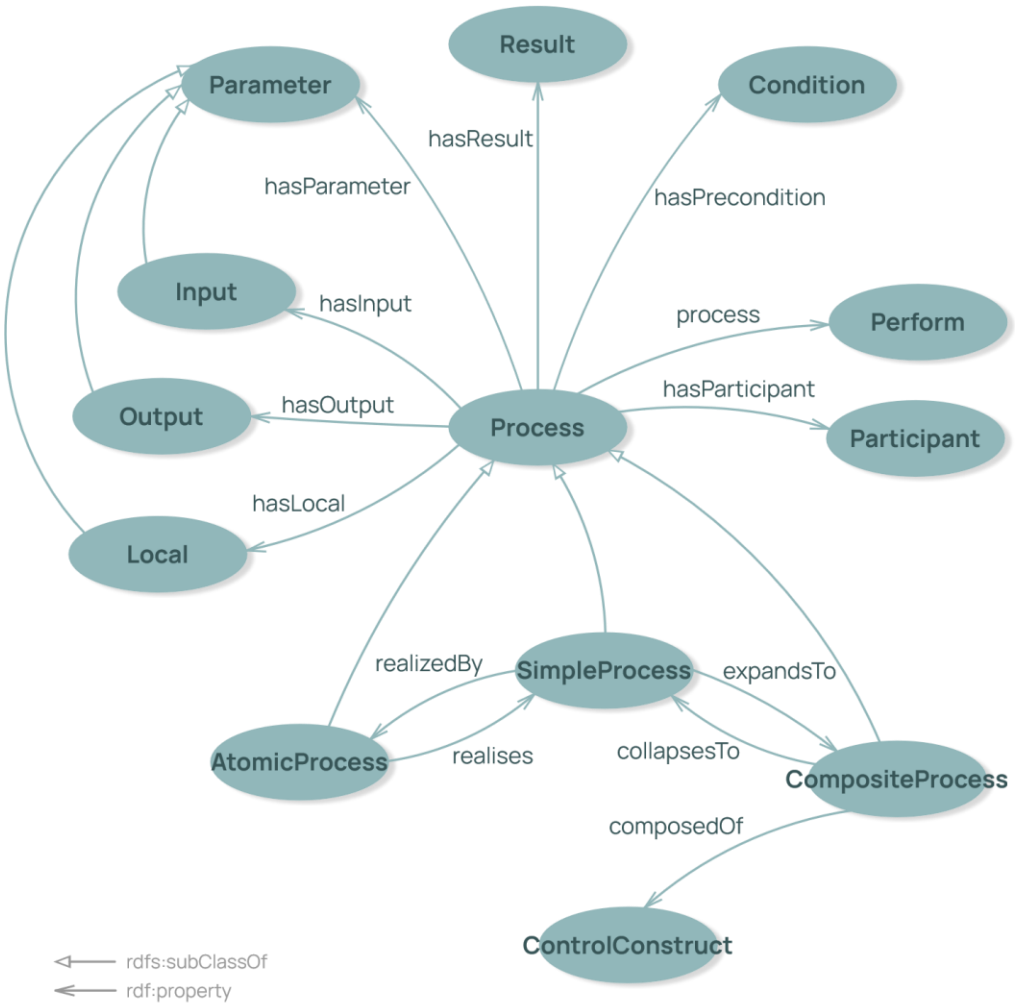


Figure 3.11
Ontology Web Language for Services selected main classes and properties (modified from W3C, 2004a)

The Function Ontology (FNO) is a lightweight ontology inspired by the OWL-S that describes signatures of the functions. Functions in the context of FNO are processes performing a specific task by associating inputs to outputs (De Meester et al., 2023). The ontology is technology independent and can support development of web services in any physical implementation. To do so, it includes three complementary vocabularies to describe mappings, implementations and compositions of function concepts to actual code implementation. While main classes in the ontology are *fno:Function*, *fno:Parameter*, *fno:Output* and *fno:Execution*, connected with *fno:executes*, *fno:expects* and *fno:returns* (Figure 3.12), these are linked to classes in complementary vocabularies to allow creation of web services. For example, *fno:Execution* links function's parameter placeholder with

the actual data in RDF, and *fnom:Mapping* maps this placeholder to corresponding argument of a function in e.g. java code. The technology independent and lightweight semantic descriptions of FNO makes this ontology easily applicable in different domains of use. For this reason, it will be reused in the proposed population disaggregation ontology of this thesis.

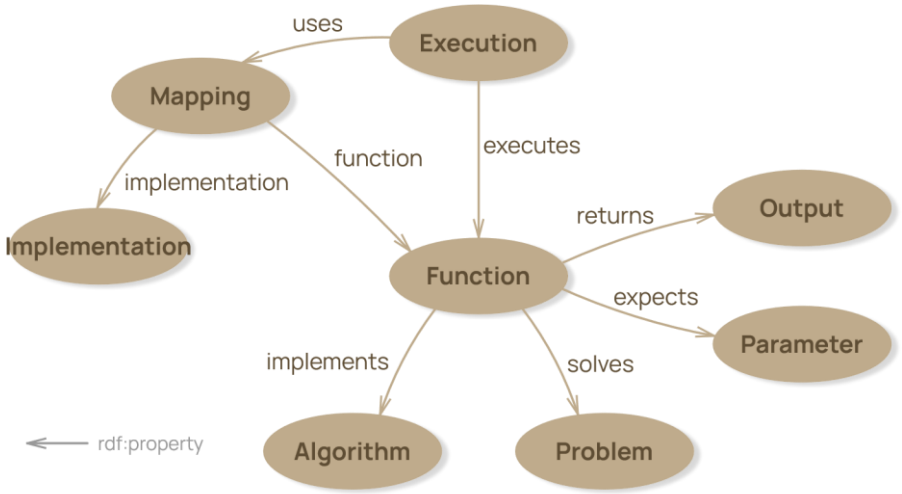


Figure 3.12
The Function ontology core classes and properties (modified from De Meester et al., 2023)

Spatiotemporal Population Disaggregation Ontology (POPDO)

When creating an ontology, there is no one correct way to model the domain of interest. Depending on the planned purpose of use and anticipated extensions to it, ontology of a domain can be modelled in viable different ways (Noy & McGuinness, 2001).

Usually, knowledge about the domain is extracted from a top-down or bottom-up conceptualization approach. Bottom-up approaches use more specific concepts of the domain to shape main classes of the ontology, while in top-down approaches the assumption is that generic underlying framework must be generated first to ensure appropriate concept integration at lower levels. But no matter the approach, Arp et al. (2015) propose eight principles for ontology creation that should ensure ontology's widespread accessibility and usability:

1. Realism. Concepts represented in the ontology should be based on general features of reality
2. Perspectivalism. Reality can be represented in different ways and all of them can be accurate
3. Fallibilisim. Reality might never be revealed in all its totality and ontologies are revisable to reflect new discoveries
4. Adequatism. Entities of reality must be considered as they are, not how they support reduction to other entities
5. Reusability. Ontologies must reuse knowledge that already exists in other relevant ontologies
6. Balance Utility and Realism. No sacrificing of realism should be made to adhere to short-term ontology usability
7. Open End. Ontology is an ongoing process: should be maintained and updated with new knowledge
8. Low-hanging Fruit. Intuitive concepts are defined first and used to define more complex one

When creating an ontology, two main inputs should be considered: new ontology must be built on existing ontologies and should appropriately model domain of interest. Noy &

McGuinness (2001) propose to start the ontology creation by reflecting on four main questions that will help to define the scope of the domain modelled in ontology.

What is the domain that the ontology should represent?

Determining the domain of ontology will narrow down the number of concepts that should be considered. By focusing on specific concepts, it will be easier to determine representative classes and relations between them. Also, clear domain definition will help to keep the focus during creation, which is highly appreciated during dilemma situations. Scope of the domain can also be determined using competency questions, i.e. questions ontology is set to answer. These will help to filter out classes of interest.

What is the intended use of the ontology?

Intended use of the ontology can help to define the level of detail in class description. Depending on the users, ontology should include assertions that are user relevant. For example, if the user of ontology is data scientist, ontology should include information about data license. The aim of this question is to make ontology relevant for its intended users.

What question the ontology should provide answers to?

To find the most suitable domain model, competency questions will reveal how classes should be connected in the ontology. This will ensure that domain of interest, user needs and the purpose of the ontology are fully aligned.

Who will use and maintain the ontology?

Answering this question will affect how ontology should be documented and explained. For example, if the ontology is written in language that user of the ontology is not familiar with, ontology should include mappings between these languages. Also, if the creator of the ontology will not maintain it, ontology must be well documented to support future upgrades.

Following these questions, main inputs for the creation of spatiotemporal population disaggregation ontology can be drawn. To start with, the ontology must model spatiotemporal population disaggregation, which is a process occurring on statistical data tied to spatial boundaries. This limits the scope of the ontology to modelling how these

data interrelate and the way they are used within the process. Also, statistical data is a variable of time so time consideration must also be given attention here. Ontology is intended to be used within automated web services that will perform the disaggregation on user demand which means the ontology must be suitable for implementation in web service technologies. Further on, as the main purpose of spatial disaggregation is to estimate more accurate spatial distribution of population, the ontology should answer a simple question of how many people lives within user area of interest. Finally, the ontology is to be used by ontology designers and web developers so the descriptions provided should primarily meet their needs.

4.1 Population Disaggregation Ontology (POPDO)

Given the discussion in chapter 3.6 (Semantic Web and Population Disaggregation) spatiotemporal population disaggregation ontology (POPDO) is an application ontology that reuses concepts from spatial, statistical and time domain ontologies and links them to semantical descriptions of tasks to form semantically enriched application in the domain of interest. To achieve this, POPDO can be divided into three layers that model how actual data is to be used in the process. For clarity, these layers are named POPDOd (domain), POPDOr (role) and POPDOp (process) (Figure 4.1).

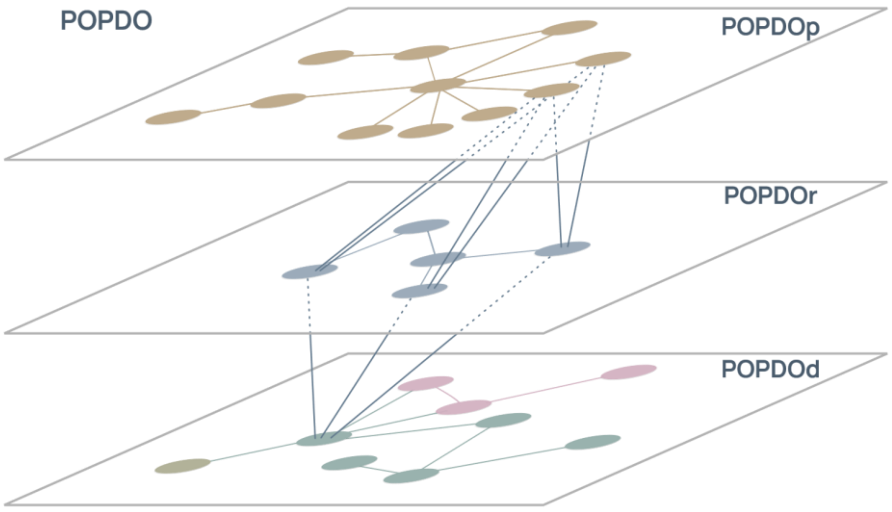


Figure 4.1
Three-layer POPDO ontology architecture. POPDOd captures data representation concepts (domain), POPDOr describe roles for data from POPDOd and POPDOp accesses data in POPDOd via POPDOr roles

4.1.1 POPDO domain layer (POPDOd)

POPDOd is the domain part of the POPDO ontology that describes how existing domain ontologies are connected into a meaningful model suitable for interconnecting data relevant in the disaggregation process. Starting with the idea previously described in chapter 2.2.1 (Population Disaggregation), it's the spatial component of population data that allows application of spatial disaggregation techniques in population disaggregation process. For this reason, geospatial domain is put in the core of the model and time reference, and statistical data are treated as its attributes (Figure 4.2).

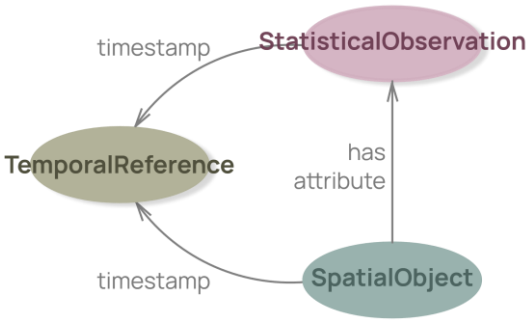


Figure 4.2
Conceptual representation of POPDO domain layer. Classes and properties are highly abstract concepts representing basic domains and interdomain relations in POPDOd.

Figure 4.2 illustrates the main integration approach within POPDOd in which classes from different domains form a uniform ontology model. This model is the basis for data representation in POPDO.

Forming the core of POPDOd, geospatial domain includes four main classes to represent spatial objects (Figure 4.3). *SpatialObject* is the master class with the main purpose of this class being to represent a concept of any kind of object that can be spatially identified. By making this class non-specific and highly conceptual, model will enable unrestricted integration of concepts at different levels of domain ontologies included. For this reason, this class is considered not to have direct instances.

The proposed definition of the *SpatialObject* class meets the description of *SpatialObject* class from the GeoSPARQL vocabulary (namespace *geo*). To keep the POPDOd model clean from semantical overload while maintaining intended purpose of the class,

SpatialObject is treated as equivalent class of *geo:SpatialObject*, i.e. *geo:SpatialObject* is reused as a class in POPDOd of POPDO ontology.

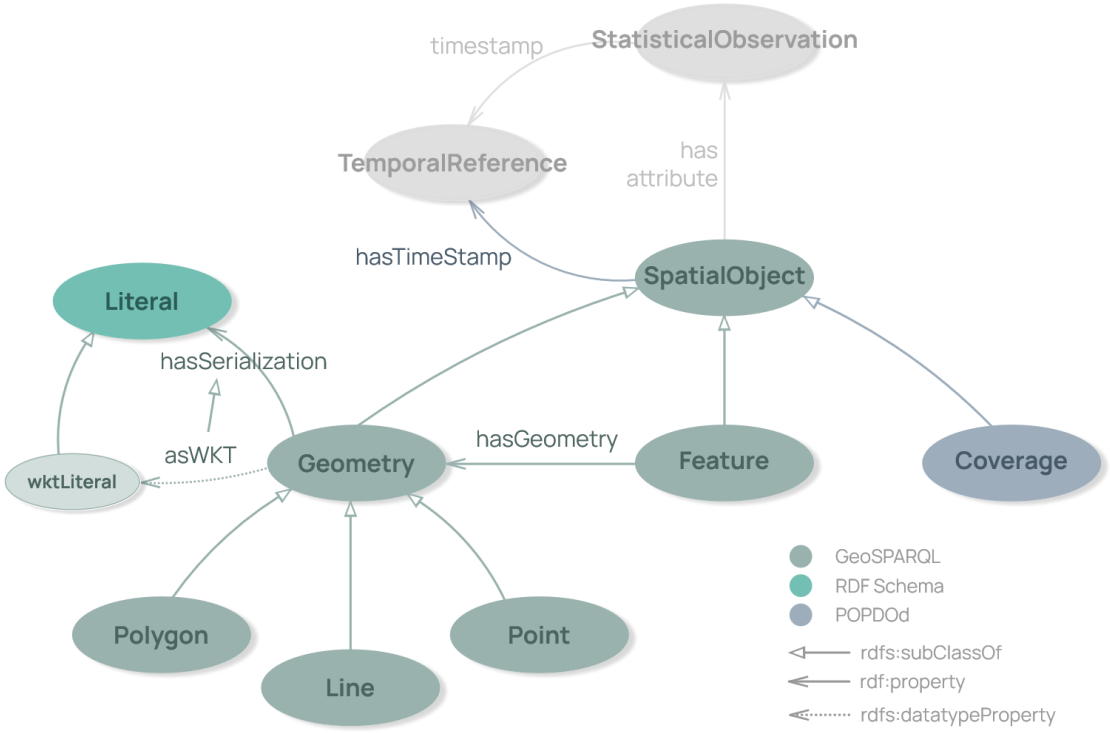


Figure 4.3
Base classes and properties representing geospatial data in POPDOd

As discussed in chapter 2.2.3 (Method using ancillary data), process of population disaggregation is based on a variety of geospatial data which can be structured in different forms. Within the geospatial community, main data forms are vector and raster so to make ontology consider their differences, and consequently to support wide ontology applicability, *SpatialObject* class is specialized through its three main subclasses.

Representation of vector data in POPDOd model is proposed with two classes, *Feature* and *Geometry*, both subclasses of *SpatialObject.* While *Feature* represents a concept of discreet object that is easily identified in space, its boundary and spatial extent is considered within *Geometry*. This dual representation of the same object allows to make semantical distinction between what an object is and how it is represented in geometrical sense and tends to add flexibility while using the proposed conceptualization model. Also, this allows to make more specific statements about the object. For example, it is possible

to differentiate between point, line and polygon representation of vector geometry (Figure 4.3), which is specifically beneficial in the realm of population disaggregation.

Description of *Feature* and *Geometry* classes in POPDOd align with the description of *Feature* and *Geometry* classes in the GeoSPARQL vocabulary, and again, these are reused in POPDOd rather than semantically reproduced. Such reuse not only reduces semantical redundancy but also enriches POPDO with existing relations that hold in GeoSPARQL. For example, features can be linked to their geometry using an existing *geo:hasGeometry* property, geometry can be serialized using well-established data types (e.g. geo:*WktLiteral*) built on top of recommended geospatial standards (e.g. Simple Features), and queries over spatial data can employ spatial functions that are not contained in the basis of SPARQL language.

Following, *Coverage* class is introduced to represent raster data used in the disaggregation process. Unlike *Feature* and *Geometry*, *Coverage* is introduced as a high-level class. This is mainly because of lack of raster representation ontologies. However, its high-level abstraction allows it to easily expand to other concepts in raster representation domain ontologies. As it is used for the definition of geospatial data, this class is made subclass of *SpatialObject* class.

Data from geospatial domain serve as carriers of statistical information about population and it allows contextualization and appropriate interpretation of represented phenomena. Within POPDOd this information is used to model connection between statistical data and geospatial ontology (Figure 4.2Figure 4.4). Observation class of RDF Data Cube vocabulary (QB) is used to describe instances of statistical data by describing them with three kinds of statements to provide context. Dimension statements, as described in chapter 3.7.2, define circumstances of the observation, or in this case referent spatial and/or temporal extents.

In POPDOd, *Observation* class description complies with the semantical interpretation of *Observation* class in QB (namespace *qb*). For this reason, *qb:Observation* is introduced as a reference class for statistical data in POPDOd (Figure 4.4). By reusing its semantics, statistical data will be given full context in statistical domain (structure and consistency), and a way to be linked to concepts from other related domains.

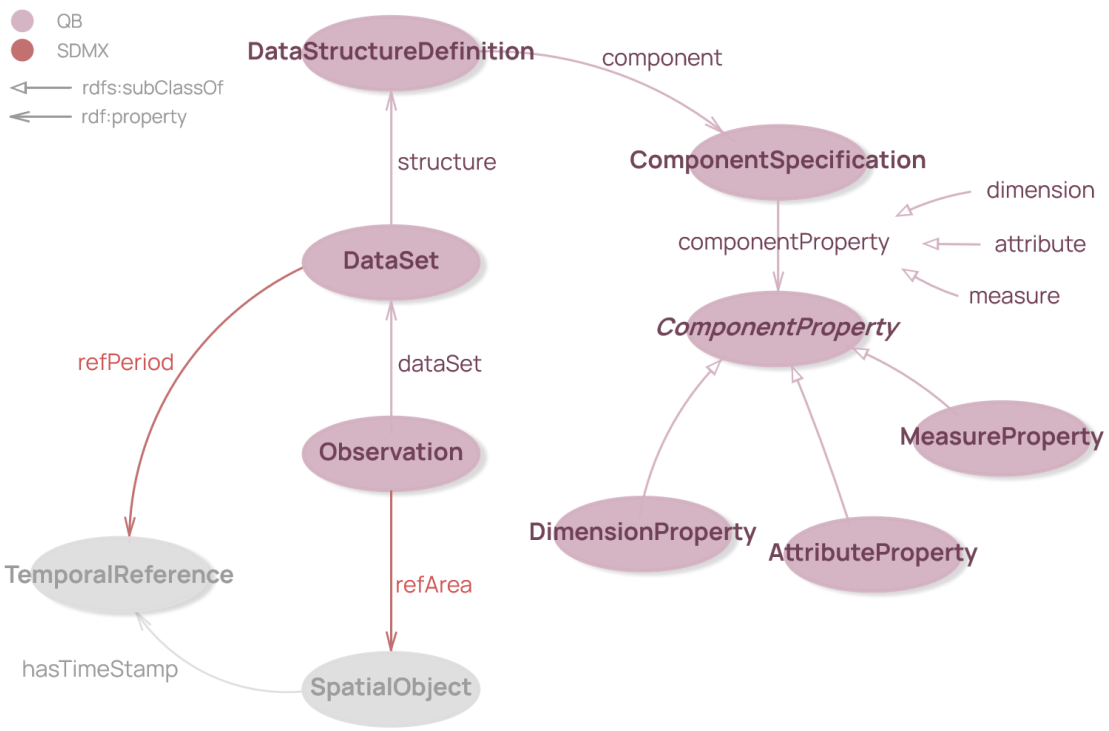


Figure 4.4
Base classes and properties representing population (statistical) data in POPDOd

Although covering statistical domain, POPDOd only uses relevant classes and properties from QB (Figure 4.4). This does not limit interoperability as data can still be linked to other concepts in the QB but keeps POPDO ontology clean and concise.

Within QB, *qb:DataStructureDefinition* defines how *qb:Observation* will be structured. Here, object properties *qb:dimension*, *qb:attribute* and *qb:measure* of *qb:ComponentSpecification* class point to their corresponding specific properties that will be used to link actual data to structure definition. These specific properties are individuals of *qb:DimensionProperty*, *qb:AttributeProperty* and *qb:MeasureProperty* classes. POPDOd considers two *qb:DimensionProperty* individuals for statistical data, namely reference area (*refArea*) and reference period (*refPeriod*) for spatial and temporal contexts. To keep the good practice, these individuals are reused from SDMX-dimension vocabulary (namespace sdmx-dimension) and not reinvented. SDMX-dimension describes *sdmx-dimension:refArea* as property that points to geographic area statistical data relates to while *sdmx-dimension:refPeriod* points to instant or period of time

statistical data refers to (United Nations Department of Economic and Social Affairs, 2019).

In QB, *qb:DataSet* class groups observations based on a common characteristic. As single population dataset usually contains multiple statistical records tied to different geographical regions, common characteristic for the dataset is time reference. For this reason, *sdmx-dimension:refArea* is introduced in POPDOd as object property that links *qb:Observation* to *geo:SpatialObject*, while *sdmx-dimension:refPeriod* links *qb:DataSet* to concept of time (Figure 4.4).

Given that proposed disaggregation model considers time as attribute of population and geospatial data, time is introduced in POPDOd as *Instant* class from OWL Time ontology (namespace time) (Figure 4.5). The class is intended to add time reference for spatial and population data, which is needed for temporal adjustment of population in the disaggregation process. As previously described, POPDOd links statistical and time domain ontologies using *sdmx-dimension:refPeriod* object property of *qb:DataSet* and it introduces *hasTimeStamp* object property to link *geo:SpatialObject* to its time stamp (Figure 4.5).

Given that the POPDO model considers timestamps for temporal part of the modelling, the model introduces years as time references. Within the model, years are introduced as individuals of *xsd:gYear* datatype from XML Schema Definition Language and linked to *time:Instant* using *time:inXSDgYear* datatype property. XML Schema is a W3C recommendation that describes primitive literal datatypes (e.g. string, double) within semantic web. To ensure POPDO is interoperable, these datatypes and data properties are again reused from existing vocabularies and ontologies (XSD Schema and OWL-TIME ontology), and not user created.

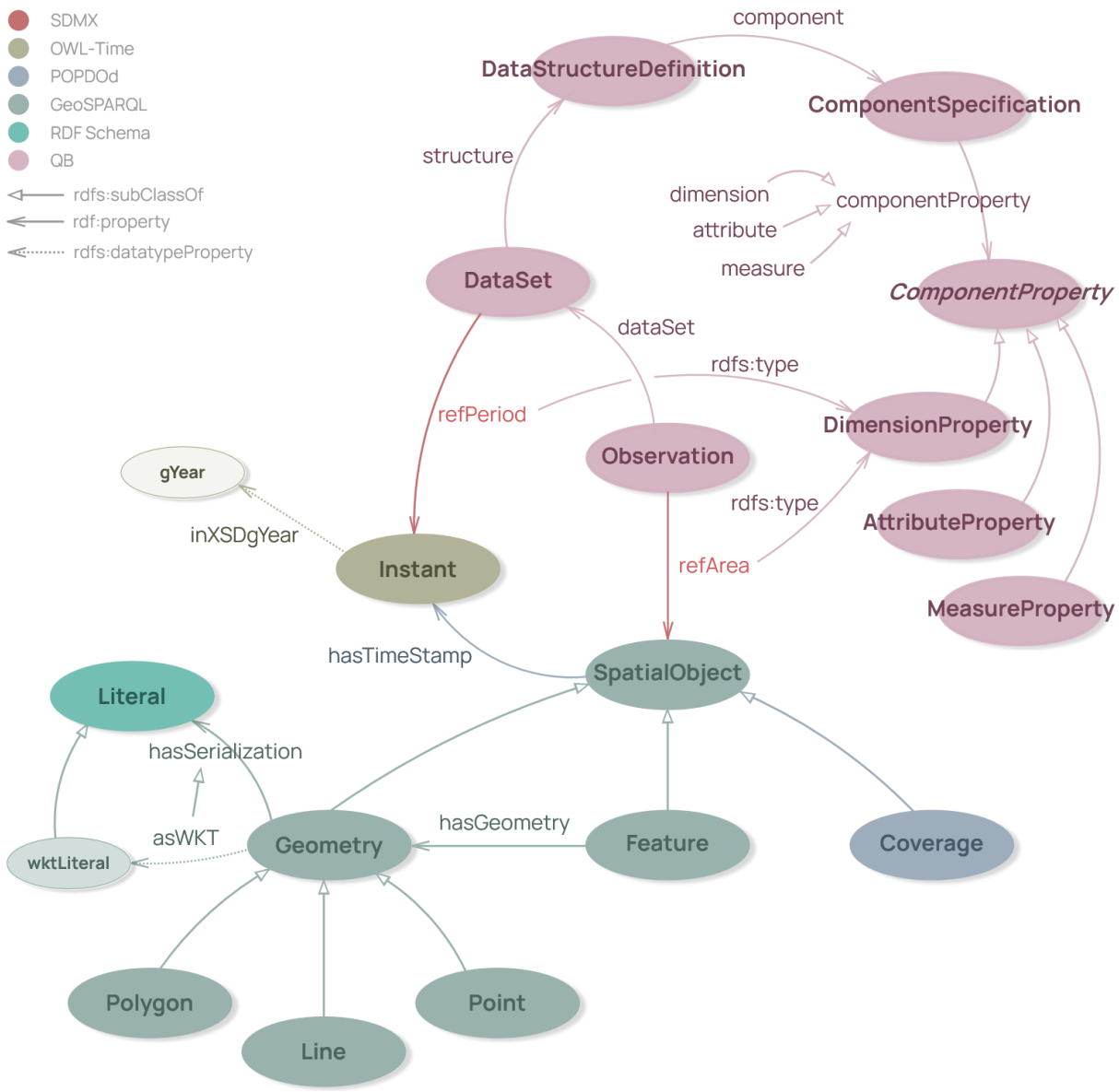


Figure 4.5
Core classes and properties of POPDOd domain layer

4.1.2 POPDO role model (POPDOr)

As discussed in chapter 2.2.4 population disaggregation approaches all share the same conceptualization: methods disaggregate statistical data from source zone to target zone(s) using either geometrical characteristics of source zone or spatial ancillary data (Figure 4.6).

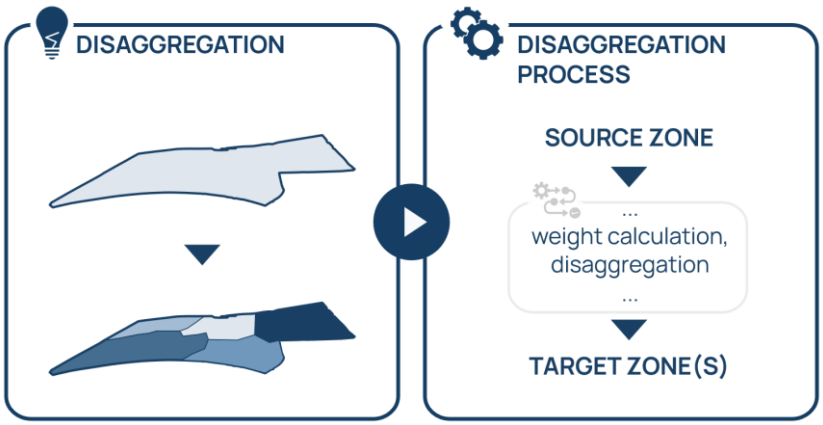


Figure 4.6
General approach to population disaggregation using spatial disaggregation methods

Within the context of population disaggregation, source zone and target zones are areas of (un)known population, while ancillary data is any kind of geospatial data that can inform about spatial presence of population. When linked to entities of the real world, these concepts may represent different things. For example, source zone can be administrative or statistical unit and although *fiat* (human agreed upon) objects, these are usually considered tangible, with precise boundary and an area. To reflect the real nature of concepts within the model, POPDOr classes are not considered as actual data but rather as roles geospatial data take within the disaggregation process. These classes and their relations are represented in Figure 4.7.

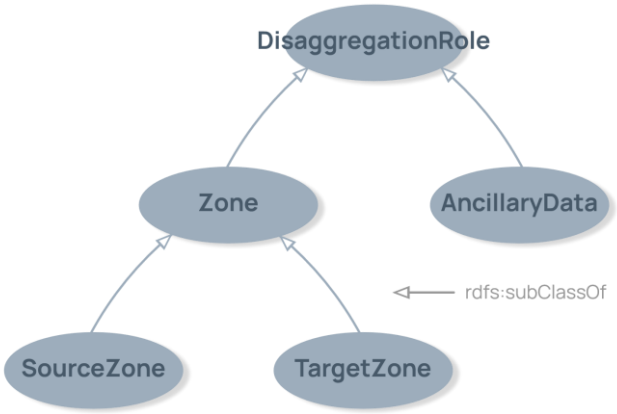


Figure 4.7
Core classes and properties of POPDOr role layer

Semantical description of role classes in Figure 4.7 should be done thoughtfully as disaggregation ontology must be universally applicable across different disaggregation methods. To start with, general concept in POPDOr is *DisaggregationRole.* This class is meta class that represents all kinds of roles data can have in the process. As such, *DisaggregationRole* is not considered to have individuals.

Roles in the disaggregation process can be divided into two major types: *Zone* and *AncillaryData*. Within POPDOr this is formalized by making these classes subclass of *DisaggregationRole*. Zones represent areas which disaggregation occurs at and are a direct linking to intrinsic components of the disaggregation method. However, to make a clear conceptual distinction between what is the input in the disaggregation process and what is the output, these zones are split into *SourceZone* and *TargetZone* classes. Again, both are considered types of *Zone* and are made its subclasses. Finally, AncillaryData class represents additional data that could help determine weights for population distribution during the disaggregation process.

POPDOr classes are role classes which means they do not accommodate raw geospatial data. Instead, geospatial data from POPDOd become individuals of these classes if they meet requirements of roles in the disaggregation process. These requirements are imposed by semantical restrictions in the description of every POPDOr class.

Source zone in the spatial disaggregation process is considered any spatial unit that has population attribute. In the POPDOd, relation between population data and geospatial

data is defined by *sdmx-dimension:refArea* object property. However, this property is directional, meaning it has domain of *qb:Observation* and range of *geo:SpatialObject*. To meet the requirement of source zone, POPDOd is extended with *isRefAreaFor* property oriented in the opposite direction. Instead of introducing *isRefAreaFor* as new and fixed property, dependency between these two is modelled using owl:inverseOf. This means that reasoner will read *isRefAreaFor* is inverse of *dmx-dimension:refArea* and infer domain of the former is the range of latter and vice versa. However, this new property makes spatial unit a candidate for source zone role, but it does not make it a source zone. To make a clear statement of what is source zone, restriction axioms in class definition were added. These restrictions defined *SourceZone* as a role class of all individuals that are both *geo:SpatialObject* and serve as reference area for at least one observation individual via *isRefAreaFor* property. Since source zone is a prerequisite for spatial disaggregation, POPDO considers even more strict semantical definition - it defines *SourceZone* as a defined class (*owl:equivalentTo*):

$$\text{SourceZone} \sqsubseteq \text{DisaggregationRole}$$

$$\text{SourceZone} \equiv \text{SpatialObject} \sqcap \exists \text{isRefAreaFor.Observation}$$

This means that every individual of *SourceZone* is individual of *geo:SpatialObject* (is geospatial object) that serves as reference area for observation, but it also means that every *geo:SpatialObject* that serves as reference area for observation is a source zone. This strict definition will allow reasoner to automatically classify spatial objects with population data attribute as source zone for the disaggregation process.

The definition of the target zone role is less restrictive than that of the source zone, primarily due to the nature of the concept in the disaggregation process. While the source zone is a prerequisite for disaggregation and possesses a clearly defined meaning, the target zone simply refers to the area to which data is disaggregated, and it must not coincide with the source zone. Consequently, the target zone role can be defined as a subclass of *geo:SpatialObject* that is disjoint from the source zone class:

$$\text{TargetZone} \sqsubseteq \text{geo:SpatialObject}$$

$$\text{TargetZone} \sqcap \text{SourceZone} \sqsubseteq \bot$$

AncillaryData role class also has less restrictive description than SourceZone class. Any kind of geospatial data that is used in disaggregation process can be defined as ancillary data as long as it is not source zone. This distinction from source zone arises from the fact that if no ancillary data exists, disaggregation is performed using characteristics of the source zone, and these are not considered ancillary data. Semantical description of AncillaryData role class axioms can be defined as follows:

$$\text{AncillaryData} \sqsubseteq \text{geo:SpatialObject}$$

$$\text{AncillaryData} \sqcap \text{SourceZone} \sqsubseteq \bot$$

Restrictions in class definition make it clear for reasoner what the class looks like. Without explicitly stating these, data could have multiple roles within the same disaggregation process which is not always the case. In other words, if something is not stated, it does not mean it is false. This comes important when considering semantical description of *AncillaryData* in respect to *TargetZone*. According to the definition of ancillary data in the context of population disaggregation, the same geospatial dataset can serve dual purposes: it can act as ancillary data to improve the disaggregation results, and can function as target zones for the disaggregated population, such as land use classes. To accommodate this flexibility, no disjoint restriction is imposed between the *AncillaryData* and *TargetZone* classes. While the main condition is fulfilled, data being individuals of *geo:SpatialObject*, it could be both ancillary data and target zone at the same time.

Figure 4.8 illustrates Class axiom graph. Nodes of this graph represent role classes from POPDOr and edges represent logical axioms that define these classes.

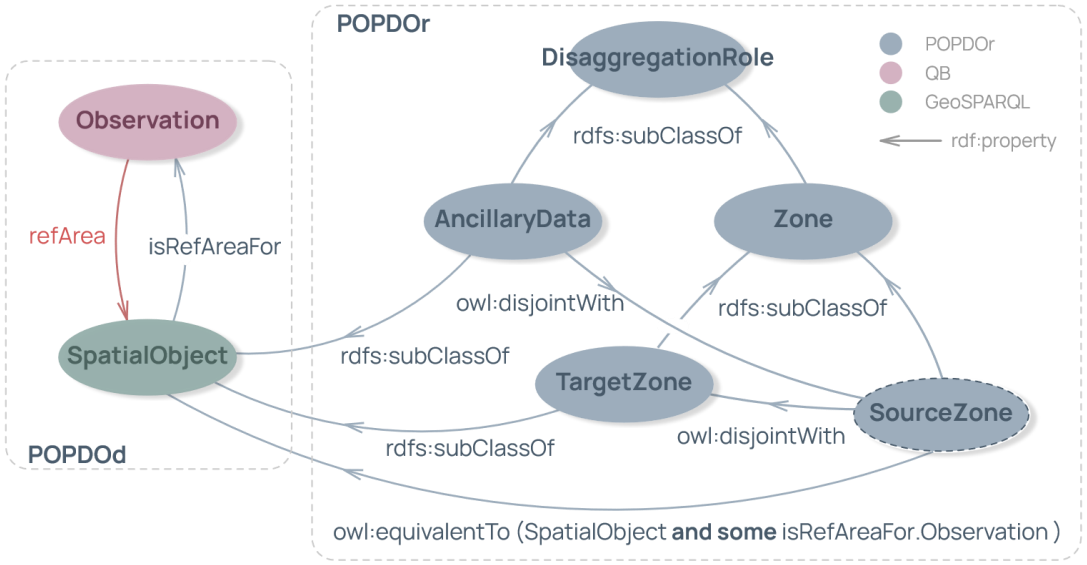


Figure 4.8
Axiom graph for POPDOr classes based on POPDOd concepts. Dashed class represents *defined* class

4.1.3 POPDO process model (POPDOp)

In chapter 2.2.4 disaggregation process is described as a procedure in which spatial weights are determined and applied to aggregated population counts to create better spatial distribution of population. This review informs about main procedure steps that occur during the disaggregation process and can be illustrated as follows (Figure 4.9):

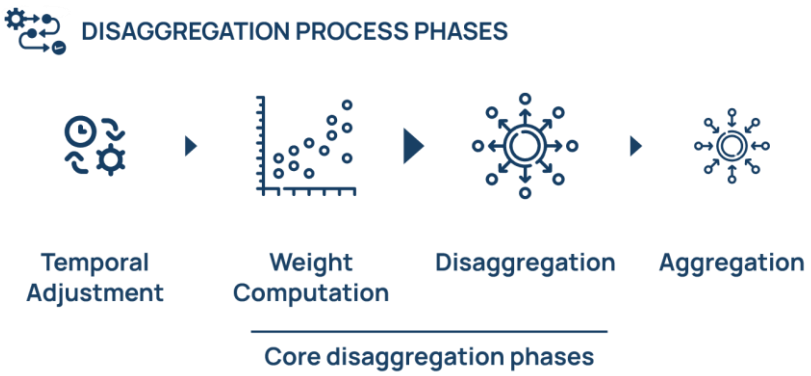


Figure 4.9
Description of POPDO as four-phase disaggregation process: core disaggregation phases extended with population temporal adjustment and aggregation phases

Disaggregation method can be described using weight computation and disaggregation phases, which is why these two phases form the core of POPDOp model. But, since disaggregation can be considered wider process than this, these two phases are sometimes

not sufficient to describe all included steps. For example, population data is highly impacted by its time reference and in many cases it must be manually updated before entering the disaggregation process. Further on, there also may exist a need to determine a total number of people living within a spatial unit of arbitrary spatial boundaries. If these boundaries do not coincide with used geospatial ancillary data, disaggregation must be done on smaller spatial units whose population values are then aggregated to resolution of the arbitrary unit. Because of this, POPDOp extends core disaggregation phases with two additional phases, temporal adjustment and aggregation, which makes it suitable to describe more comprehensive disaggregation processes (Figure 4.9). These phases are therefore referred to as disaggregation steps.

Each disaggregation step represents a group of procedures whose goal is to perform calculations and ensure disaggregation step's result. Temporal adjustment, for example, includes temporal update procedures that result in more accurate population data at source zone. Weight computation conceptually represents a group of procedures that result in spatial weights needed in the disaggregation process, and disaggregation describes procedural steps that lead to disaggregated population data. At last, aggregation performs aggregation procedure over disaggregated data to indicate population counts within arbitrary spatial boundaries. As these procedures perform actual computations, within POPDOp they are considered algorithm steps.

Arising from requirements of general population disaggregation method, POPDOp main classes should represent conceptual groupings where each grouping accommodates more domain-specific procedures. These procedures work as a pipeline that define how specific part of the process is to be executed to produce result. In a broader context, this allows to describe disaggregation method as groups of nested procedures within overall disaggregation process.

POPDOp model within POPDO is based on The Function Ontology (FNO) described in section 3.7.4 and extended to accommodate specificities of population disaggregation process. The FNO is a lightweight task ontology intended to describe processes in technology independent way, and it offers comprehensive semantical descriptions

applicable not only to procedures as individual steps, but also to linking of these into an executable pipeline.

The *fno:Function* is the central class of FNO (namespace fno) and POPDOp model (Figure 4.10). It represents any kind of process where input is linked to output making it a highly abstract concept universally applicable. To model a process, *fno:Function* establishes relations to classes representing input and output via standard object properties, *fno:expects* and *fno:returns*. While conceptually these properties link functions to inputs and outputs, formally, FNO ontology defines ranges of these properties to be of type *rdf:List* (Figure 4.10). By doing so, the *fno:Function* class is allowed to have a list of and not a single input/output value. Inputs and outputs in FNO are described with *fno:Parameter* and *fno:Output* classes. Both classes represent only concepts and can be interpreted as semantical descriptions of data placeholders rather than explicit data.

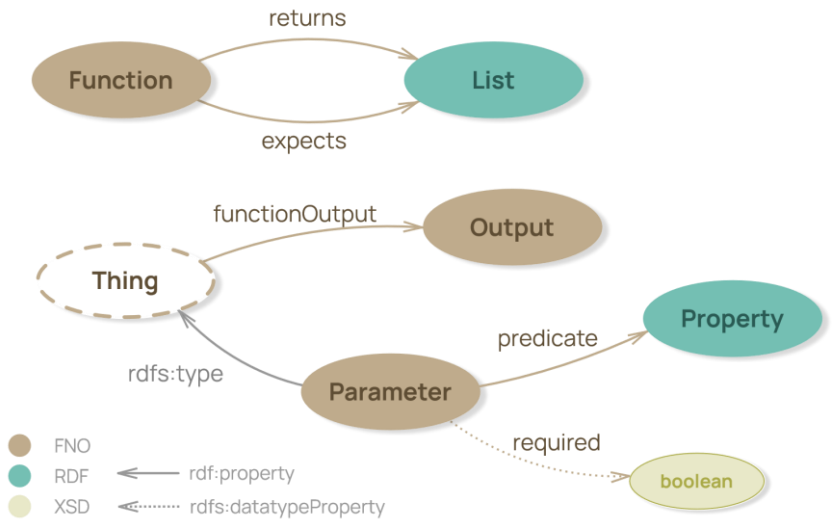


Figure 4.10
Core process description classes and properties of FNO ontology reused in POPDOp (modified from De Meester et al., 2023)

To describe data within *fno:Parameter* placeholder, FNO uses three main properties. The *fno:type* property defines what kind of data, i.e. data literal or individual of another class placeholder can take as input value. The *fno:required* allows to mandate existence of data within the placeholder. If the value (Boolean) is true, the parameter must receive data and cannot stay empty. Finally, the *fno:predicate* property makes it possible to link the placeholder to actual data values. However, it only describes how this can be done, and

not how it is actually done. For this reason, the range of *fno:predicate* is *rdf:Property* class, i.e. another property. This new property is then used by the executor class to store values of a specific type in the placeholder during implementation.

These three main classes of FNO are enough to semantically describe the bare act of processing. However, only semantical description of the process in general terms is not sufficient for its execution. This requires additional classes that will model how real data are placed into placeholders of *fno:Parameter* and *fno:Output*, and finally, how these are connected and used within concrete implementations.

To enable execution of procedures, FNO introduces *fno:Execution* class (Figure 4.11). Main purpose of this class is to establish semantical connection between real data in Semantic Web (RDF) and their corresponding placeholders in FNO. To do so, it uses RDF object (property) of *fno:predicate* as its own data property. The *fno:Execution* acts upon *fno:Function*, formally using *fno:executes* property and it serves as interface class to enter the process (function execution) from outside of the semantic web during implementation.

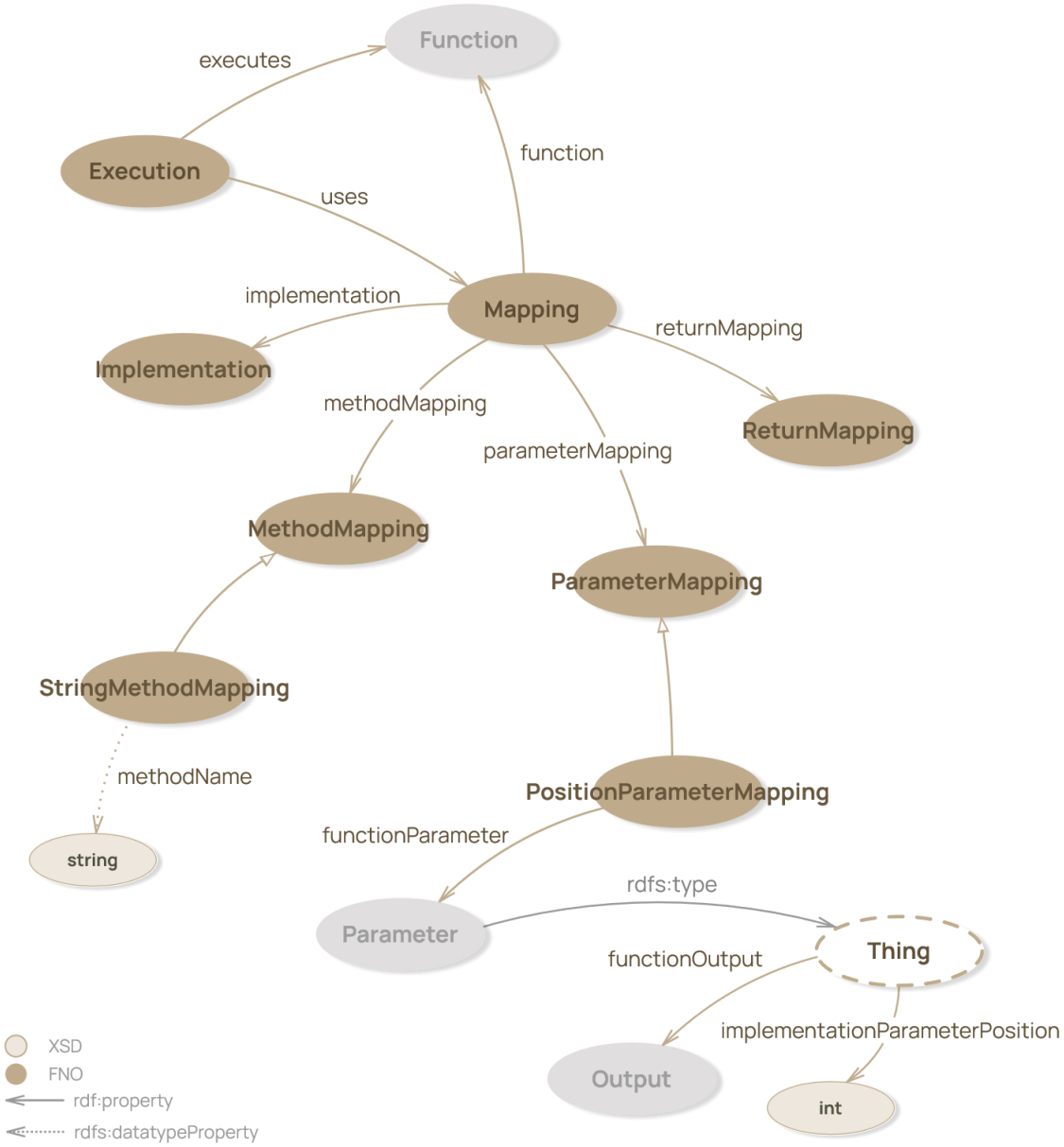


Figure 4.11
Selected classes and properties of FNO ontology needed for the execution of a function in POPDOp (modified from De Meester et al., 2023)

Even though *fno:Execution* adds important piece of knowledge to description of process, this is still not sufficient for most of real-world applications. While execution of the process can indeed happen using SPARQL query language, but, as its name state, this is a query language meaning it has limited possibilities to support procedural nature of the process. For this reason, new piece of knowledge needs to be added to align semantical descriptions of ontology with more powerful execution technologies like Java. Within FNO, this gap is bridged with two additional classes, *fno:Mapping* and *fno:Implementation*, that model how

functions, based on their semantical descriptions of inputs and parameters, are implemented outside of the Semantic Web (Figure 4.11).

The *fno:Implementation* class provides semantical descriptions of practical implementation of executions, usually linking to external platforms outside of the realm of Semantic Web. It gives instructions for the implementation framework of semantic descriptions. Instances of this class usually point to pieces of programme code developed for specific calculation problems.

To make function description applicable to external implementations, it is needed to reconcile descriptions in ontology with their meanings in external technologies. For example, individual of *fno:Parameter* has the semantic interpretation of argument in Python method. In FNO, this linking is done with *fno:Mapping* class that, again, semantically describes how ontology concepts are used in implementation instances (Figure 4.11). The *fno:Mapping* uses *fno:function* property to link to *fno:Function* and *fno:implementation* property to link to *fno:Implementation* class. By using *fno:parameterMapping*, *fno:methodMapping* and *fno:returnMapping* properties, individuals of *fno:Mapping* unambiguously map parameters and outputs of function individuals to method's arguments in external execution scripts, often by their name or position in method definition. This way, a connection between function's parameter and output in ontology is linked to its counterpart in execution technology.

As sometimes execution of one function requires other functions to be executed too, FNO introduces the possibility of a composition. Composition in FNO is considered alternative to implementation described above and defines semantically how parameters and outputs of functions are connected in a flow (Figure 4.12). The *fno:Composition* class defines what the composition is and how it looks like. Using *fnoc:composedOf* property, it establishes a connection with *fnoc:CompositionMapping* that maps how parameters and outputs of one function are forwarded to the next function (*fnoc:mappingFrom* and *fnoc:mappingTo* properties). Finally, *fnoc:CompositionMappingEndpoint* defines which functions and their parameters and outputs are used in this process (*fnoc:constituentFunction*, *fnoc:functionOutput*, *fnoc:functionParameter* properties).

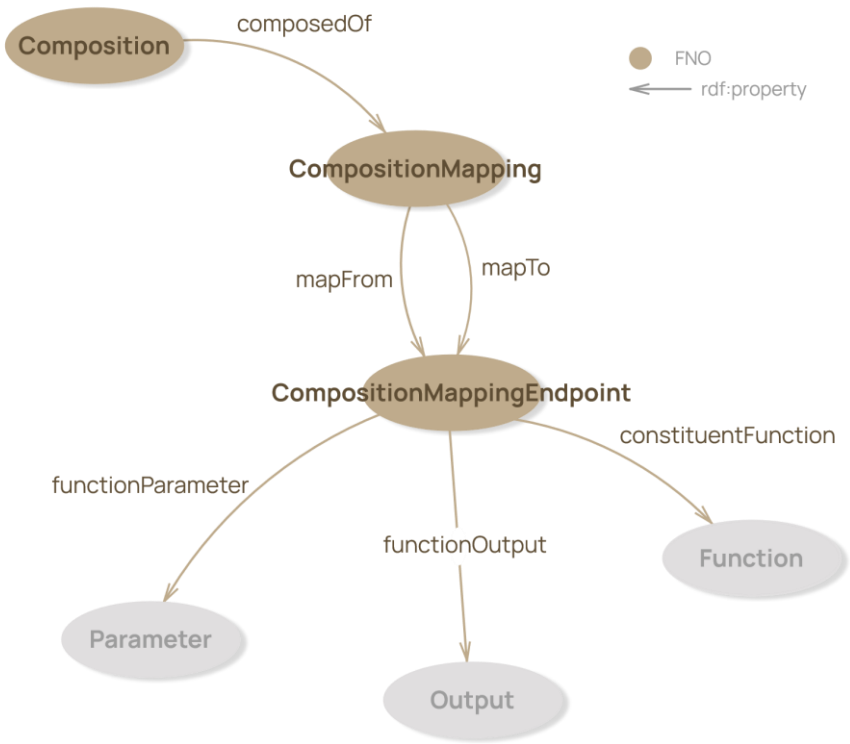


Figure 4.12
Selected composition-related classes and properties from FNO ontology within POPDOps (modified from De Meester et al., 2023)

While reusing concepts from FNO in POPDOp model, the nested characteristic of the disaggregation process had to be preserved which raised a challenge of how to fit POPDOp semantics into a semantical framework of FNO.

To achieve fully automated population disaggregation process in POPDO by reusing FNO knowledge, several statements about FNO should be drawn:

1. *fno:Function* is highly abstract class defined only as a process that links inputs to outputs,
2. *fno:Execution* populates parameters and outputs of only one function, i.e. every *fno:Function* individual has its own *fno:Execution* individual,
3. *fno:Execution* individual links to only one individual of *fno:Mapping*, meaning it can only guide execution of one *fno:Function* individual,
4. *fno:Composition* works with *fno:Functions* and is alternative to implementation.

Given the requirements of the population disaggregation methods and its execution, the following requirements need to be met:

1. Population disaggregation should be automated, i.e. it should take aggregated data and produce disaggregated data.

   This means that population disaggregation method should be of type *fno:Function* so that *fno:Execution* instance can populate its parameters with actual data (source zone, ancillary data) and define the output (target zone)

2. Disaggregation process (method) is a sequence of disaggregation steps performing specific parts of the calculation.

   This means that every step has an input that is either source zone (and ancillary data) or output from the previous step. This leads to conclusion that every group should be of type *fno:Function* so that *fno:Execution* instance can map these values into correct placeholders.

3. Disaggregation steps contain one or more procedures where each one does atomic calculations.

   This means that every procedure that does calculation needs to have specified input and output. Therefore, every procedure should be individual of *fno:Function*.

Fully functional and universally applicable population disaggregation process should be automated and scalable in execution, i.e. functions should be chained to ensure data flow. If every concept of population disaggregation process is simply treated as function of FNO and chained into a composition, semantical descriptions of procedures would be either very generic or very detailed but lacky in comprehensiveness. In other words, disaggregation method would be described as only one generic function for the entire process, or as a sequence of individual procedures which are not tied under the single disaggregation method. On the other side, if combination of these different level semantical expressivities is tried, this could break the process logic and potentially end in inexecutable process. To avoid such problems, proposal in POPDOp is to create subclasses of *fno:Function* that will represent three levels of procedures in the disaggregation process (Figure 4.13). *DisaggregationMethod* class represents the entire disaggregation method.

For individual of this class, *fno:Execution* individual defines parameters and output, i.e. source zone and target zone in the disaggregation process. *DisaggregationStep* class represents phases of the disaggregation process and exploits potential of composition to get input from disaggregation method level. *AlgorithmStep* class accommodates procedures of disaggregation step performing calculations. By being *fno:Function*, this class will reuse composition expressivity to get input from disaggregation step, and via implementation it will be linked to external scripts that will perform calculations.

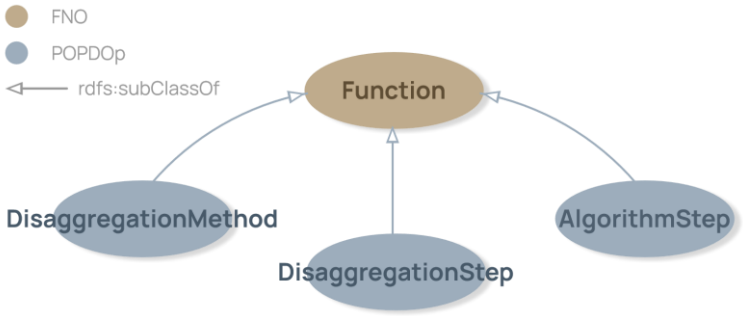


Figure 4.13
POPDO procedure classes as subclasses of fno:Function

New classes of POPDOp, introduced as subclasses of *fno:Function*, add the procedure-level knowledge into the semantical model and allow creation of modular disaggregation process. By doing so, all parts of the POPDOp meet the requirements of population disaggregation process while adhering to expressivity of FNO model. Structuring the model this way, it ensures that population disaggregation method, as top level procedure, has input source zone and output of type target zone, while the rest of the execution workflow is based on semantical description contained in composition and implementation individuals.

Proposed spatiotemporal population disaggregation ontology is a top-level domain ontology in the realm of population disaggregation. While it considers general concepts that hold in the domain, its practical application requires refinements to meet the needs of a specific disaggregation method. Entire POPDO model with its three-layer architecture can be found it appendices section (Appendix A).

Automated Population Disaggregation

Within this chapter, application of the POPDO ontological model in practical disaggregation is described. For proof of concept in this thesis, general POPDO ontology is conceptually extended to support disaggregation based on Building Area Dasymetric Mapping method (BDASY).

Testing of the ontology was made on Trogir city administration unit (Croatia), using official statistical data (census population counts and population yearly changes) from Croatia National Bureau of Statistics and official geospatial data (administrative unit boundaries and building footprints) provided by the State Geodetic Administration. To test the ontology, several steps were conducted (Figure 5.1). First, to suit the needs of application, data required by the BDASY method were retrieved and preprocessed in external software. Secondly, concepts in the ontology were formally structured, i.e. serialized for automated processing by the computer using Protégé, an open-source ontology-creation software. Following, schema individuals and data individuals were structured in RDF data model while adhering to concepts from POPDO. Next, ontology and RDF triples were merged into a meaningful knowledge graph in GraphDB graphical database compliant with W3C standards. At last, automated execution disaggregation of population data was showcased in a local environment using predefined python scripts.

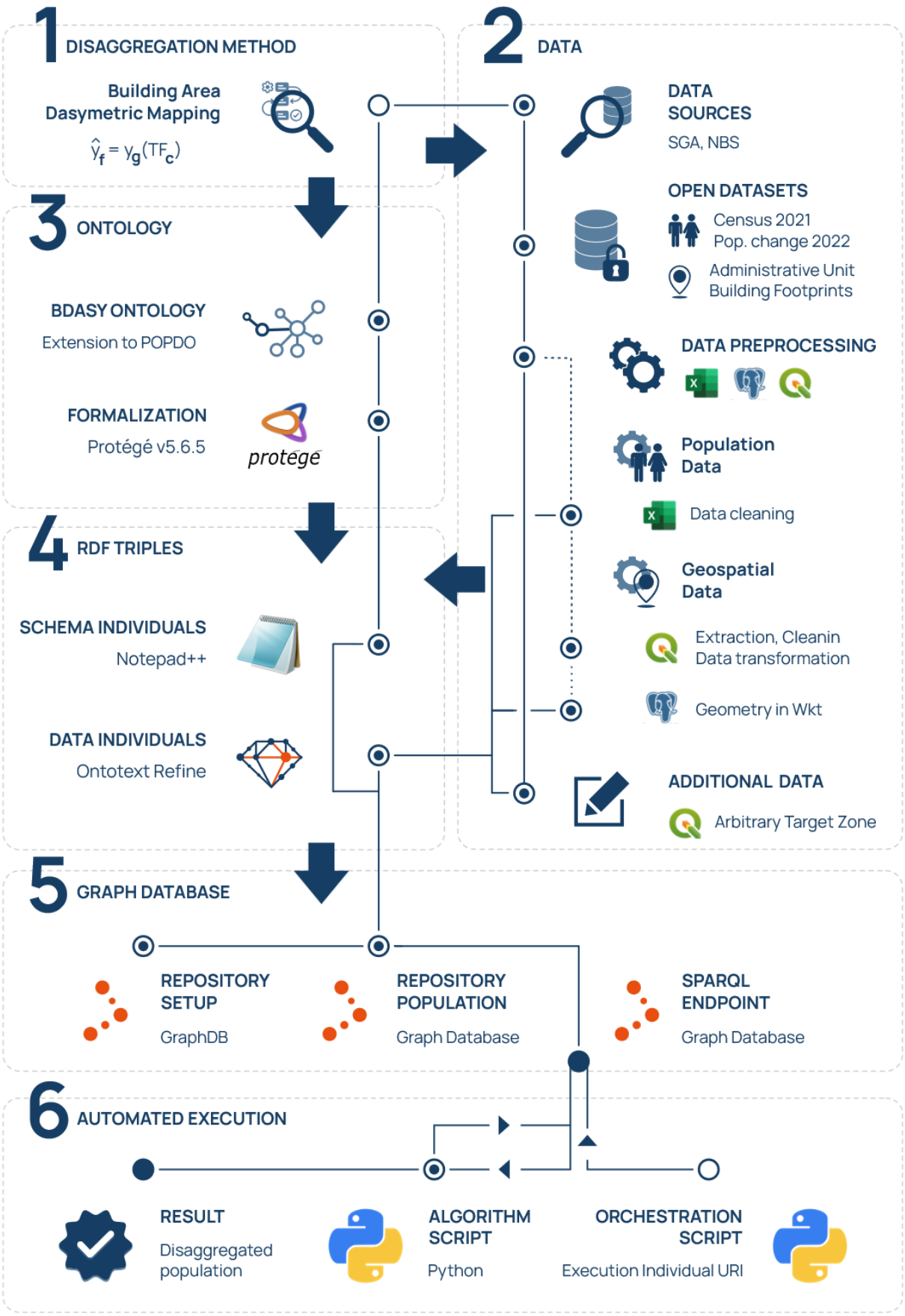


Figure 5.1
Population disaggregation testing workflow

5.1 Setting the Agenda

5.1.1 Building Area Dasymetric Mapping Method

Based on research by Sapena et al. (2022) which suggests methods using building footprints as ancillary data tend to provide more accurate results and given the fact that this data in Croatia is provided as open data by official sources, dasymetric mapping using building footprints was decided to be the core disaggregation model. For simplicity of showcasing, the method adopted builds on top of the core model by considering only area of buildings.

Building Area Dasymetric Mapping (BDASY) is a type of dasymetric mapping where building footprints are used as main ancillary data to develop spatial weights in the disaggregation process. Besides disaggregation, the process was extended to temporally adjust population data from census reference year to year of interest, and to give results aggregated to arbitrary spatial unit.

Disaggregation algorithm in BDASY is based on modified approach proposed by Mennis (2015). This algorithm uses area ratio and population density ratio to calculate total fraction at the level of residential buildings which is then used as weight in the disaggregation of population. More precisely,

$$\hat{y}_f = y_g \left(\frac{AR_f \, DR_c}{\sum_{f \in g} (AR_f \, DR_c)} \right)$$

$$AR_f = \frac{A_f}{\sum_n A_f},$$

$$DR_c = \frac{\widehat{D}_c}{\sum \widehat{D}_c},$$

$$\widehat{D}_c = \frac{\sum_n y_f}{\sum_n A_f},$$

where $\hat{y}_f$ is the estimated disaggregated population for every residential building in the source zone, y_g is the population of the entire source zone, A_f is the area of a residential building, AR_f is the area ratio per building per residential class (if more than one), $\widehat{D}_c$ is

estimated population density of a residential building class (if more than one), DR_c is population density ratio per residential building class (if more than one).

5.1.2 Data Sources

Identified by the DBASY, disaggregation process seeks for administrative unit spatial representation for the source zone, building footprints polygons for ancillary data and statistical data on population counts and yearly population changes. All these datasets in Croatia are provided as open data by government sources.

Population Data

Data on population in Croatia is produced and maintained by the National Bureau of Statistics. It is an integral part of Census of Population, Households, and Dwellings which is conducted with decennial temporal resolution. The most recent census is the one conducted in 2021 based on traditional approach of enumeration. Official time reference for the census is 31 August 2021 (Official Gazette of The Republic of Croatia, 2021).

Scope of the Census along with methodology is defined by The Act on The Census of Population, Households and Dwellings in the Republic of Croatia in 2021, which is aligned with relevant European regulations (EC 763/2008 and EC 2017/712). Census collects 36 variables about population (e.g. age, gender). These are disseminated in profiles per spatial unit, in aggregated form and with person as unit of measure. National Bureau provides the data in tabular form in .xlsx file format or in 1km x 1km vector grid (Kević & Kuveždić Divjak, 2025). Although enumeration unit is the finest spatial resolution for the dissemination, its data are only available upon request. The finest spatial unit level of openly available population data is a settlement. All data from the census are available under Open Government License.

Within this research, population data on a level of city/municipality is used for the disaggregation. As the research only concerns total number of people living within the area, different population profiles of data in the dataset have no impact on the result and any can be used for the disaggregation purposes.

Census population data is tied to time reference of 2021 and may be obsolete. To track population changes, National Bureau of Statistics publishes natural population changes on

a year level. This data is aggregated data from official birth and death registers and given up to a level of city/municipality in .xlsx file format and under Open Government License (Croatian Bureau of Statistics, 2023). For simplicity of practical implementation, POPDO ontology disaggregation was showcased for year 2022. For this reason, natural population changes for year 2022 were used in addition to Census 2021 population count.

As defined in The Act (Official Gazette of The Republic of Croatia, 2021), Census data collection and dissemination is tied to spatial units, which are kept and mainstained by the State Geodetic Administration.

Geospatial Data

Official geospatial data on administrative units and buildings in Croatia is produced and maintained by the State Geodetic Administration (SGA). Administrative units are part of the The Register of Spatial Units and building data are maintained in Cadastre records.

Spatial units of The Register of Spatial Units are classified in eight levels. Levels are hierarchical and spatial units of lower-level fit within boundaries of the upper-level. City and municipality are the same level units and within the Register are referred to as local self-government units (Official Gazette of The Republic of Croatia, 2020). For the simplicity of meaning within the disaggregation process, these will be referred to as administrative units.

As already explained, disaggregation within this testing will be performed on the level of city/municipality, more specifically on the spatial boundaries of City of Trogir in Split-Dalmatia County. Spatial unit of interest is provided under Administrative unit dataset downloadable via ATOM service in .gml file format from SGA Geoportal. Provided data is INSPIRE compliant meaning that it aligns with conceptual rules of INSPIRE data specification (State Geodetic Administration, 2025). Every unit in this dataset is described with multiple attributes, e.g. name or type of administrative unit. Data is provided in projected coordinate system (ETRS89/LAEA, EPSG: 3035) and under Open Government License. Time reference for the data is 2025 but no records of boundary change for administrative unit was found. For this reason, it was used in the disaggregation process with time reference of 2022.

Data on buildings in the disaggregation process were retrieved from Cadastre records, also maintained by the SGA. The data is INSPIRE compliant and available via ATOM service in .gml file format. As originally maintained by the Cadastre, buildings are provided on a level of cadastre units which are lower-level units when compared to city. Every building in the dataset has, among others, has unique identifier and use code attribute. Use codes identify the intended use of the building (e.g. residential, industrial) and are of relevance for the disaggregation process. Data is provided in projected coordinate system (HTRS96/TM, EPSG: 3765) and under Open Government License. As data is retrieved via atoms service, its time reference is of more recent date than the reference year of the disaggregation process. This, however, is not a concern as the main purpose of the testing is the proof of concept of automated disaggregation using semantics in ontology definition.

5.2 BDASY Formal Representation

Two main steps in creation of BDASY ontology can be distinguished. In the first step, POPDO ontology is extended with new classes representing concepts specific to BDASY, while the second step refers to how this knowledge is formalized in digital environment.

5.2.1 BDASY: extension to POPDOd

To accommodate method specific data, geospatial domain of POPDOd is extended with five main classes (Figure 5.2). The *SpatialUnit* class is introduced as a subclass of *Feature* class, and it represents spatial units used for the dissemination of official statistical data (e.g. statistical or administrative units). For the purpose of ontology showcasing, *AdministrativeUnit* is introduced as a subclass of *SpatialUnit* to match the characteristics of population data provided by the National Bureau of Statistics. *AreaOfInterest* and *BuildingFootprint* classes are defined as subclasses of *geo:Feature*. Class *AreaOfInterest* describes arbitrary spatial unit that was used as target zone in the disaggregation process and *BuildingFootprint* class represents buildings as spatial objects. Besides being described as subclass of *geo:Feature*, this class is given datatype property *hasUseCode* which allows to attach use code attribute to its individuals in ontology. To extract only residential buildings and later use them as ancillary data, *ResidentialBuildingFootprint* class is added in the model. This class accommodates all buildings whose use code value is interpreted

as residential. Members of the class are not added manually but are rather assigned automatically by the reasoner if they meet the condition of being a building and have residential value of *use code*, i.e. *ResidentialBuildingFootprint* is a defined class in BDASY ontology.

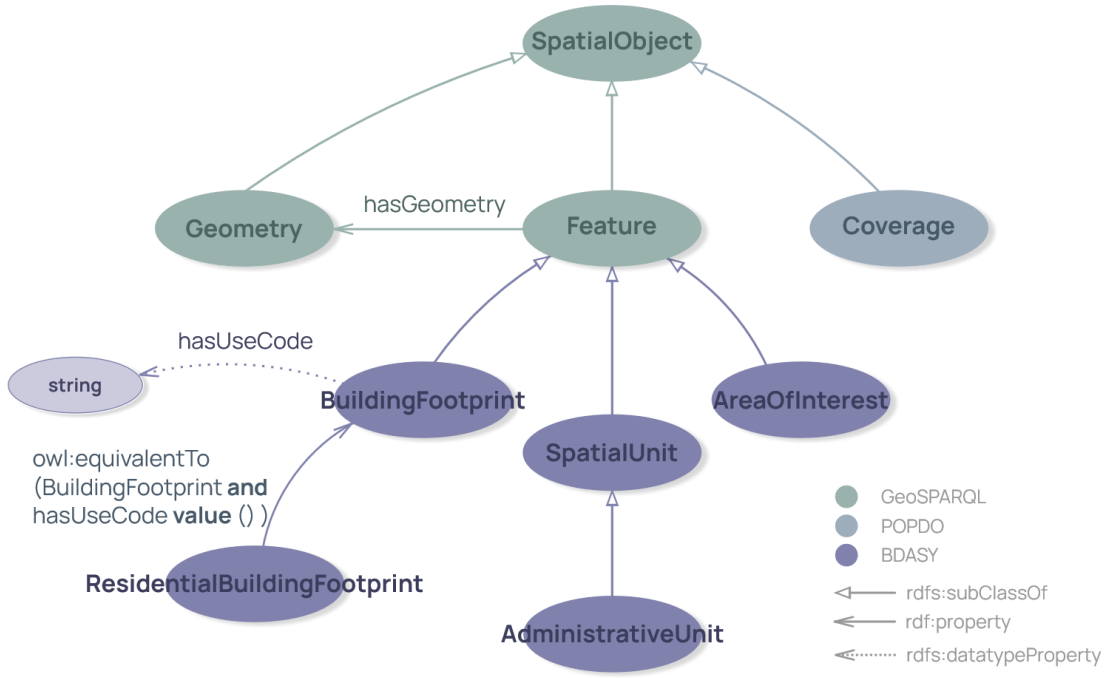


Figure 5.2
BDASY extension classes to POPDO ontology

Once modelled conceptually, BDASY ontology was formalized using Protégé to create TBox axioms for the disaggregation knowledge graph.

5.2.2 BDASY realization in Protégé

Protégé is an open-source platform for building and managing ontologies developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine (Musen, 2015). It features graphical interface that supports explicit modelling of domain knowledge using classes, properties, logical axioms and individuals based on Web Ontology Language. Creation of new ontologies can be done on top of existing ontologies which are imported from local file or from web location. Its main strengths are easy ontology creation and reasoning capabilities that assure logical consistency and inference of new knowledge. Protégé version 5.6.5 with supported ELK, HermeT, Ontop, Pellet and jcel reasoners was used for creation of BDASY ontology within this research.

Ontology creation in Protégé starts by defining new ontology template with basic information about the ontology; its URI, version, description and preferred namespace. BDASY ontology was assigned http://www.example.com/ontology/buildingareadasy/ URI with *bdasy* namespace, and this is where definition of ontology concepts will be stored. BDASY ontology builds on top of POPDO concepts, so these were imported in Protégé to form the basis. As POPDO concepts are based on domain ontologies (GeoSPARQL, OWL Time, RDF Data Cube, The Function Ontology – with its multiple vocabularies) these were also retrieved through import of POPDO URI.

Once the ontology template was in place, *AdministrativeUnit*, *AreaOfInterest* and *BuildingFootprint* classes were added in Class hierarchy as subclasses of POPDO classes and and *ResidentialBuildingFootprint* is added as subclass of *BuildingFootprint* (Figure 5.3). Classes of the active ontology are marked in bold letters to distinguish it from classes pertaining to imported ontologies. Adding a new class to Class hierarchy only adds taxonomy information. To provide more specific meaning, additional information can be assigned via Class Description module (Figure 5.4).

Classes in class hierarchy can be placed manually, so called primitive classes, or based on the axioms from the class definition. In that case reasoner infers about the class and defines its place in the hierarchy. Protégé also allows to create defined classes whose members are determined by necessary and sufficient conditions expressed in equivalent axioms.

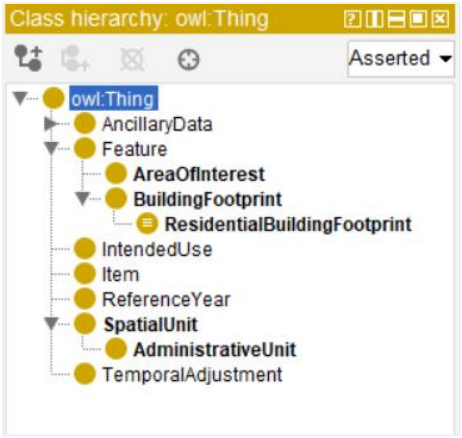


Figure 5.3
BDASY classes added as subclasses of POPDO ontology classes within Class hierarchy in Protege

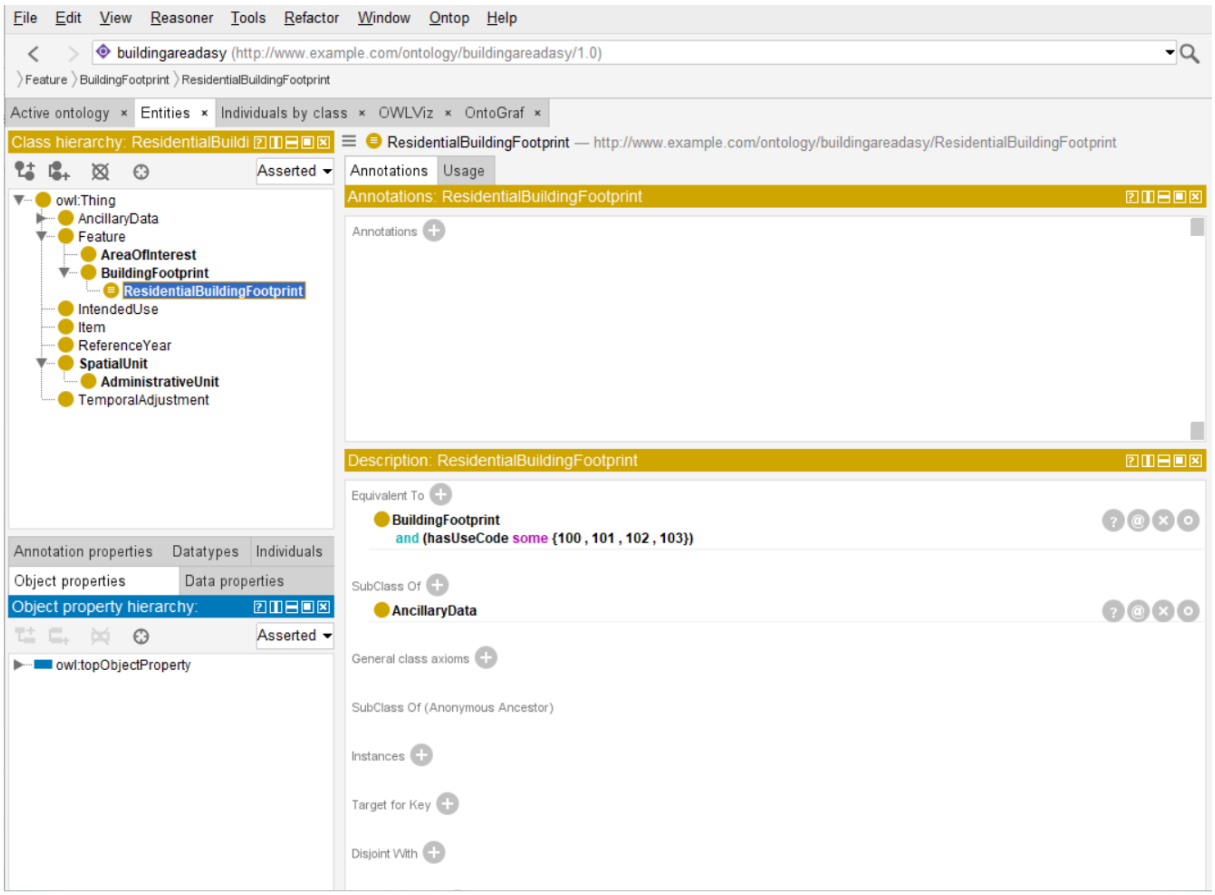


Figure 5.4
ResidentialBuildingFootprint class described as defined class within Protégé

As seen in Figure 5.4, *ResidentialBuildingFootprint* is described as a defined class whose members must be of type *BuildingFootprint* and have use code value 100, 101, 102 or 103. These values are individual-specific and come from the dataset.

Object properties establish relations between classes (and hold for their individuals) and in Protégé these are added as subclasses of *owl:topObjectProperty*, superproperty in OWL. Object properties from POPDO are sufficient for BDASY so no new object properties were added. In contrast to object properties, datatype properties link class to a literal value. Within BDASY, one new datatype property, *hasUseCode,* was added to hierarchy as a subproperty of *owl:topDataProperty* of OWL. The property was assigned domain of *BuildingFootprint* and range of string data type to constrain its semantical meaning (Figure 5.5).

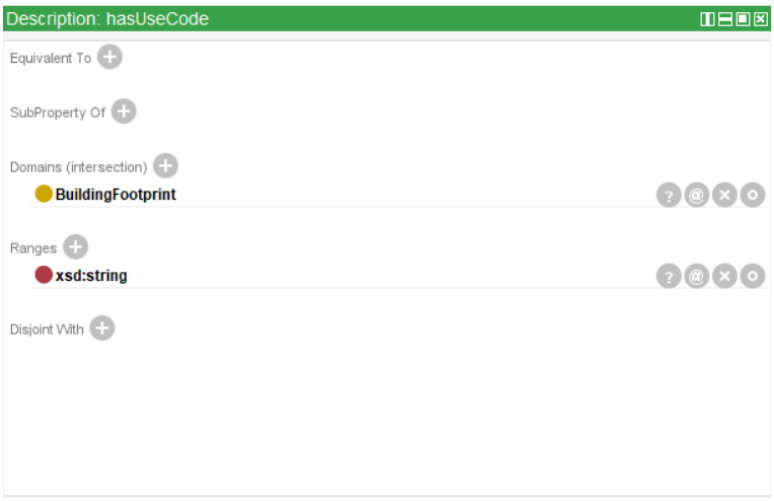


Figure 5.5
BDASY *hasUseCode* datytype property definition

Besides providing semantical axioms for knowledge representation, Protégé adds the possibility to attach annotations to every concepts. These are not used in reasoning and serve to provide more informative description in human readable style. For this reason, they are omitted from the model in this stage.

Finally, Protégé also supports capabilities of adding class individuals. This option is helpful when working with a small amount of data but can become exhausting and time consuming when dealing with large datasets like building footprints. Instead of manually inserting thousands of buildings for test area, assertions about these in the Abox of knowledge base were made in an external data manipulation software.

However, to check ontology consistency with a reasoner, several individuals were added in the Protégé. These individuals do not represent real world data and are used for testing of reasoning only.

To check if building footprints individuals are assigned to residential building footprints, and consequently to ancillary data class, two instances, *Building123* and *Building234* were created. *Building123* was attached use code 100, and *Building234* use code 200. Based on the restriction axiom *ResidentialBuildingClass*, only *Building123* should be inferred as individual of (Figure 5.6).

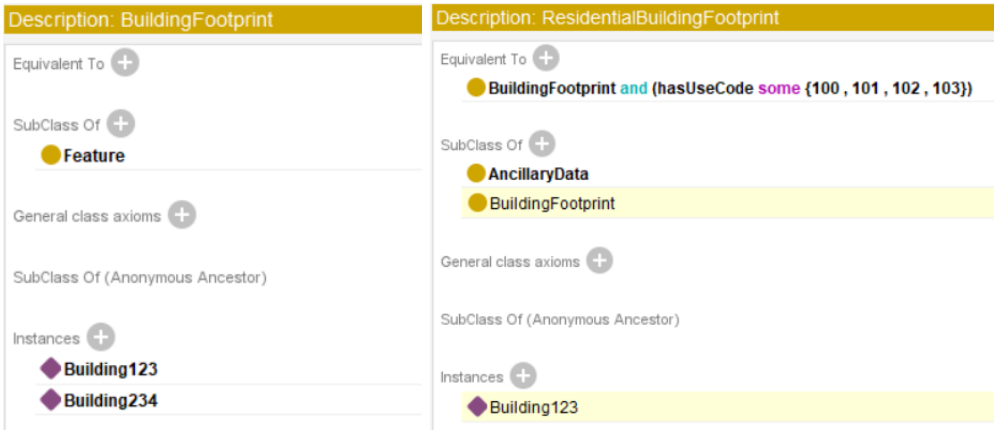


Figure 5.6
Reasoner inference results on *ResidentialBuildingFootprint* based on *BuildingFootprint* individuals

As *ResidentialBuildingFootprint* is also subclass of *AncillaryData*, *Building123* should also be inferred as its individual. Figure 5.7 shows reasoner can infer this knowledge meaning that ontology is consistent. The same test was run with both Trogir instance for Administrative unit (Source Zone role) and AreaOfInterest for AreaOfInterest class (Target Zone role) (Figure 5.8).

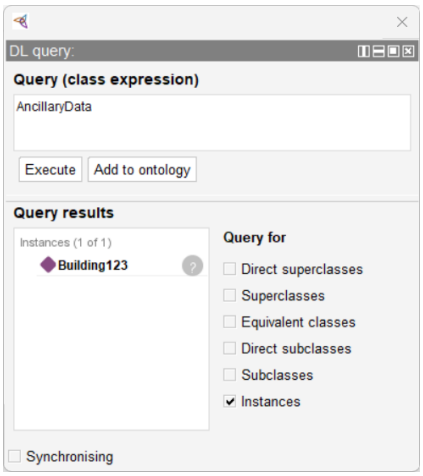


Figure 5.7
Reasoner inference on *AncillaryData* based on *ResidentialBuildingFootprint* class descipriont

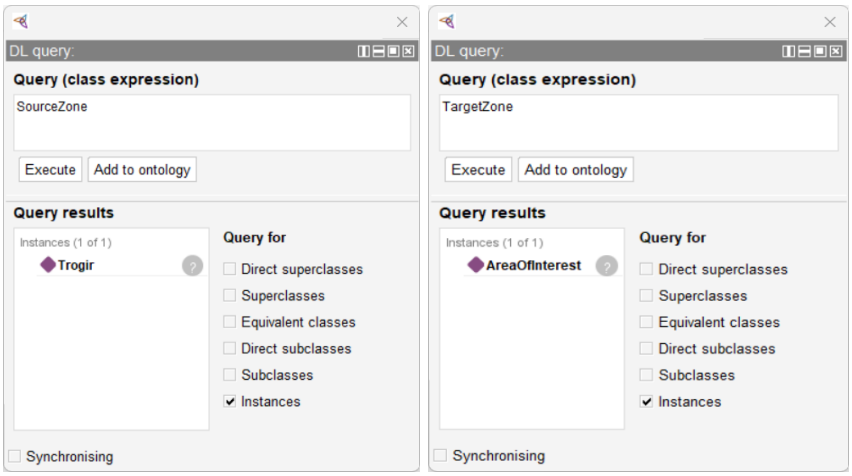


Figure 5.8
Reasoner inference on *SourceZone* and *TargetZone* based on individuals of *AdministrativeUnit* and *AreaOfInterest* classes

Reasoning on semantics in ontology showed no inconsistencies exist which means ontology is ready for the implementation.

5.3 Data in RDF

Populating ontology with real world data requires transformation of data from their native data structure into RDF data model. As previously described, this is feasible within Protégé software but can become tiring in case of large volume of data (e.g. building footprint data). For simplicity and more direct data transformation, Ontotext Refine was used for these purposes.

Ontotext Refine is a free application developed by Ontotext. It features graphical interface that allows fast cleaning, mapping and automated conversion of structured data into RDF model. It is an adoption of OpenRefine, open-source software for data curation, for working with Ontotext GraphDB graph database (Ontotext, 2022). Ontotext Refine, data curation application, and Ontotext GraphDB, graph database, are in the basis of practical implementation within this research so compliance of these tends to solve potential issues of data interoperability.

Before transforming data into RDF triples, data had to be preprocessed and structured in formats readable by Ontotext Refine. For geospatial data, QGIS (version 3.22.14 Białowieża) was used as transformation tool while population data was cleaned using Excell from MS Office 365.

5.3.1 Data Preprocessing

As mentioned earlier, Trogir administrative boundary was provided with other administrative units in a single dataset, in .gml format and in ETRS89/LAEA coordinate system. This caused several issues for intended use in disaggregation process. First, disaggregation was planned only for Trogir so the geometry of Trogir unit needed to be extracted. This was done in QGIS by filtering attribute table by the value of name in *text* column. Extracted unit was then reprojected from ETRS89/LAEA into HTRS96/TM projected coordinate system to ensure its overlap with building data. For representation in GeoSPARQL, the unit needed explicit geometry representation in Well Known Text (WKT). As QGIS has limited field lengths for attributes, representing geometry in new column of attribute table would cut out parts of WKT syntax. For this reason, spatial unit was exported from QGIS into PostgreSQL database (v17) and added geometry in WKT. Ontotext Refine offers connection to database as a form of data import so Trogir administrative unit was imported into OntotextRefine directly from PostgreSQL.

Building footprints from Cadastre were provided in separate datasets on a level of cadastre unit, so these needed to be merged. Once imported into QGIS, they were merged using *Merge vector layers* tool. For every building in the dataset, explicit geometry representation in WKT was then added as a new value in the attribute table. This was feasible as building polygons have less points in the geometry. As number of buildings exceeded 6000 objects, these were imported into Ontotext Refine via already established infrastructure of PostgreSQL database.

Data on population counts and count changes were much lighter and only needed filtering to extract relevant data. This was done in MS Excell by removing unnecessary rows and columns. Once filtered, .xlsx files were imported manually into OntotextRefine.

Disaggregation to area of interest requires this area to be provided by the user. For the testing purposes, arbitrary rectangle within Trogir administrative unit was created in QGIS, added geometry representation in WKT and imported into OntotextRefine for further modelling.

5.3.2 Source Data Individuals

OntotextRefine reads imported data and stores it into tabular structure (Figure 5.9). This form is then used to assign values to a specific poisition in RDF triple.

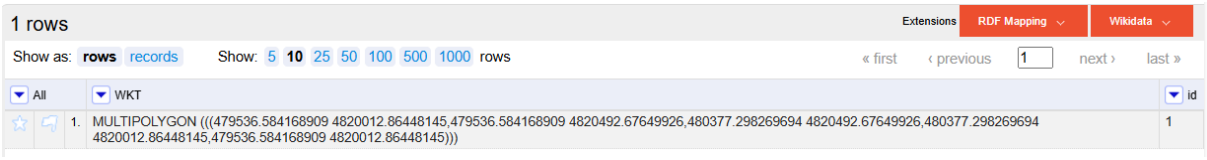


Figure 5.9
Arbitrary area of interest (target zone) representation in Ontotext Refine

RDF mapping module within Ontotext Refine is where RDF triple components are created. It consists of three main parts: base URI, qnames ribbon and triple creator. Base URI is the namespace for storing newly created triples, qnames ribbon allows to add URIs of external ontologies whose concepts are reused in newly created triples, e.g. BDASY ontology, and triple creator is where RDF triples are made from tabular data. Ontotext offers capabilities to define concepts to be constant values (e.g. fixed concept URI), dynamic values (changing across rows), or extracted from the table based on a GREL filtering rule (GREL, language for manipulation and organization of data). These capabilities of Ontotext add flexibility that is needed when creating RDF triples from tabular data.

Datasets imported into Ontotext Refine represent individuals of classes from BDASY. Therefore, these individuals should be defined within the same namespace as their respective classes. But, to ease the distinction between individuals and schema concepts, every dataset was given its own base URI:

Trogir Administrative unit **admunit:** http://www.example.com/data/admunit/

Area of interest **areai:** http://www.example.com/data/areainterest/

Building footprints **build:** http://www.example.com/data/building/

Population count **popcnt:** http://www.example.com/data/popcount/

Population change **popcng:** http://www.example.com/data/popchange/

Approach with base URIs different from ontology URI however increased the risk of error in alignment of individuals with their respective classes once imported in graph database. This is mainly because ontology concepts in Ontotext are manually defined, and

any typing error could make reasoner not to recognise triples as statements about ontology class individuals. For this reason, special attention was given to qnames definition and concepts naming. Example of how tabular data were mapped to ontology concepts during RDF triples creation is given below (Figure 5.10).

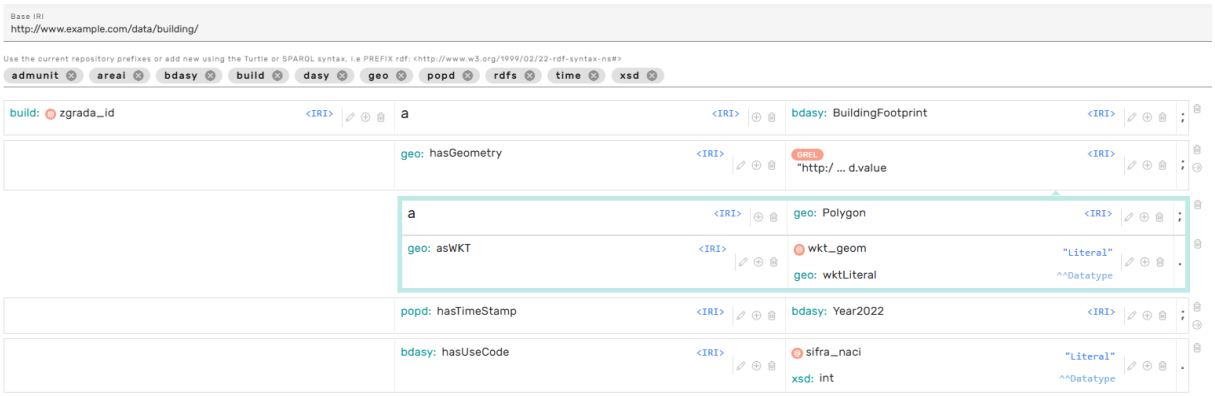


Figure 5.10
Example of RDF mapping for building footprint datas

Figure 5.10 represents the schema for mapping of building footprint data to BDASY ontology concepts. Following this schema, buildings were converted from tabular representation into RDF triples of the Semantic Web. Each row from the table was defined as individual of bdasy:BuildingFootprint class and was linked to its geometry representation in WKT, timestamp for year 2022 and assigned use code value. Once the mapping schema was completed, data was transformed in RDF and exported in .ttl file format. Given like this, it was now ready for import in graph database.

5.3.3 BDASY Schema Individuals

Along data individuals, to make the ontology fully operational, process classes must also be instantiated to embody process logic of a concrete disaggregation method. These individuals are referred to as schema individuals. But, before describing the schema and its individuals, it is necessary to understand the logic of process execution in BDASY.

BDASY ontology supports semantical descriptions of the disaggregation process. SPARQL as a query language can read these descriptions and be aware of the workflow, but it has limited procedural capacity needed for process execution. For this reason, ontology schema must be somehow linked to external technologies to perform calculations. To be more precise, BDASY allows do describe disaggregation process as

composition of compositions of functions in which only algorithm step functions (execution procedures) are implemented using external technologies (Figure 5.11). Here, the entire process is based solely on description of the workflow, not the actual data, so schema composition individuals are needed to precisely guide data flow between functions in the process.

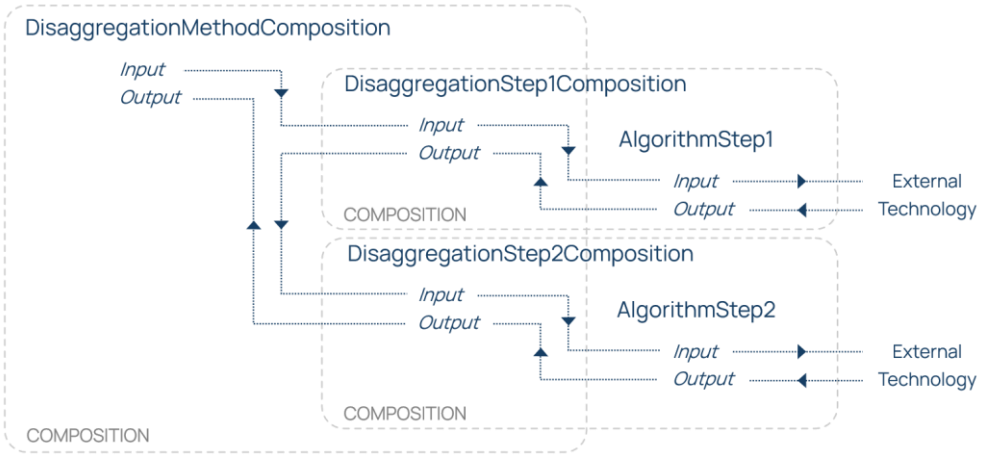


Figure 5.11
Composition of compositions as disaggregation process execution logic

Disaggregation process schema of BDASY, based on method description in section 5.1.1 and execution logic in Figure 5.11, can be illustrated as follows (Figure 5.12):

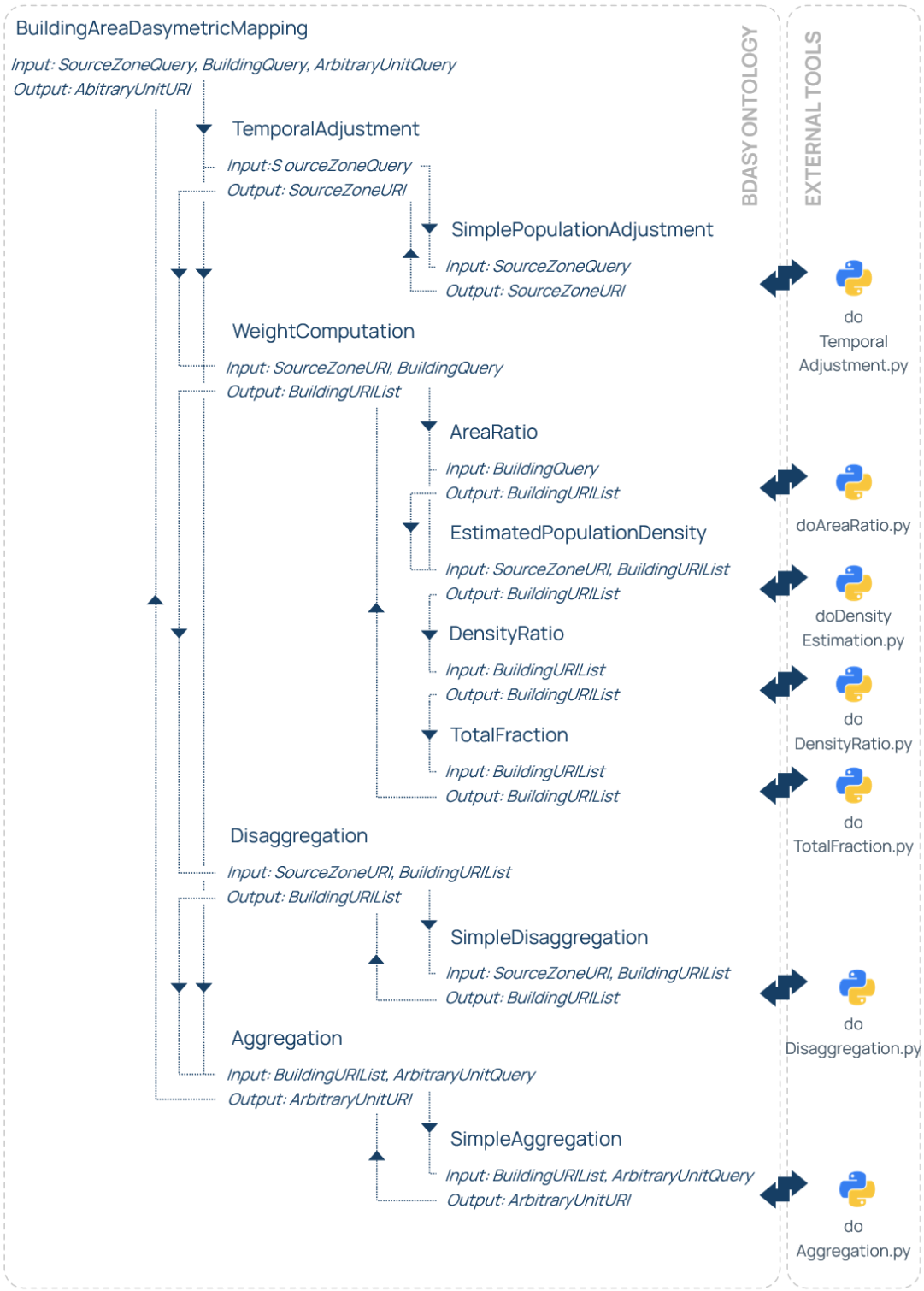


Figure 5.12
BDASY disaggregation process schema combining semantical descriptions and external execution scripts

Main components of the schema (Figure 5.12) are individuals of three-level BDASY functions. BuildingAreaDasymetricMapping, a DisaggregationMethod function

individual, is considered semantical orchestrator of the process. This individual was assigned parameter and output placeholders values. These placeholders take what is to represent source zone, target zone and ancillary data. To fill the placeholders, function corresponding Execution individual was created. To ensure process is dynamic, i.e. flexible and not fixed to specific data, instead of sourcing the function's parameter placeholders with concrete URIs of data, placeholders were given SPARQL queries in string form. This way, number of input URIs per placeholder could be arbitrary number. This is especially important for building footprint ancillary data as number of residential buildings in the dataset is not a priory known. Inputs of the disaggregation method function are then forwarded to individuals of disaggregation step functions in the composition who pass it to algorithm step functions. Individuals of Mapping class for algorithm step function link these values to individuals of Implementation class who call the python script and perform calculation. Following semantical description of output, calculated values are shared among functions.

Given that these individuals don't represent real data, but rather serve execution, they were created manually following Turtle serialization syntax. An alternative to manual creation was to define them in Protégé, but this proved to be less efficient and more time consuming. Full list and definition of schema individuals can be found in the Appendix C.

Again, to make a distinction between schema individuals and schema classes, every group of schema individuals was assigned its unique base URI:

Functions **fun:** http://www.example.com/function/

Execution **exe:** http://www.example.com/execution/

Mapping **map:** http://www.example.com/mapping/

Composition **comp:** http://www.example.com/composition/

Once all the data was ready, it was imported in graph database for automated disaggregation testing.

5.4 Automated Execution Setup

Last step before testing semantical descriptions of disaggregation process is to establish reachable graph database which will act as a server and provide data upon request. Ontotext GraphDB is a free, standards-compliant graph database. It features reasoner

capability and offers SPARQL Endpoint for access to data. GraphDB fulfils all the requirement imposed by the testing of disaggregation process, so it is chosen as technology for showcasing the BDASY ontology in practice.

Data within GraphDB is organized in repositories and every repository represents individual graph database. New repository was created for population disaggregation. To ensure reasoning capabilities, repository used OWL2 RL ruleset. Unlike Protégé whose reasoners supports full DL reasoning, GraphDB ruleset reasoning profiles are optimized for different purposes (W3C, 2012). For this reason, they have restricted expressivity suited for specific needs. OWL2 RL is a W3C recommended OWL language profile optimized for applications requiring scalable reasoning that seeks efficiency and trades off full semantic expressivity. This directly affects reasoning on BDASY ontology axioms where source zone role, defined as equivalent class of intersection with existential qualification of literal, cannot be directly populated with individuals. To overcome this limitation in testing stage, real world data was made individuals of role classes instead of domain classes. This setup allowed process execution while not altering ontology consistency proved by Protégé reasoning (section 5.2.2).

To allow OWL2 RL to make reasoning, files in GraphDB must be inserted in specific order. Ontology schema is inserted first to provide reasoning framework which is then populated by schema individuals and finally data individuals (Figure 5.13). Once imported, resourced in the graph can be visualized using Visual graph module of GraphDB (Figure 5.14).

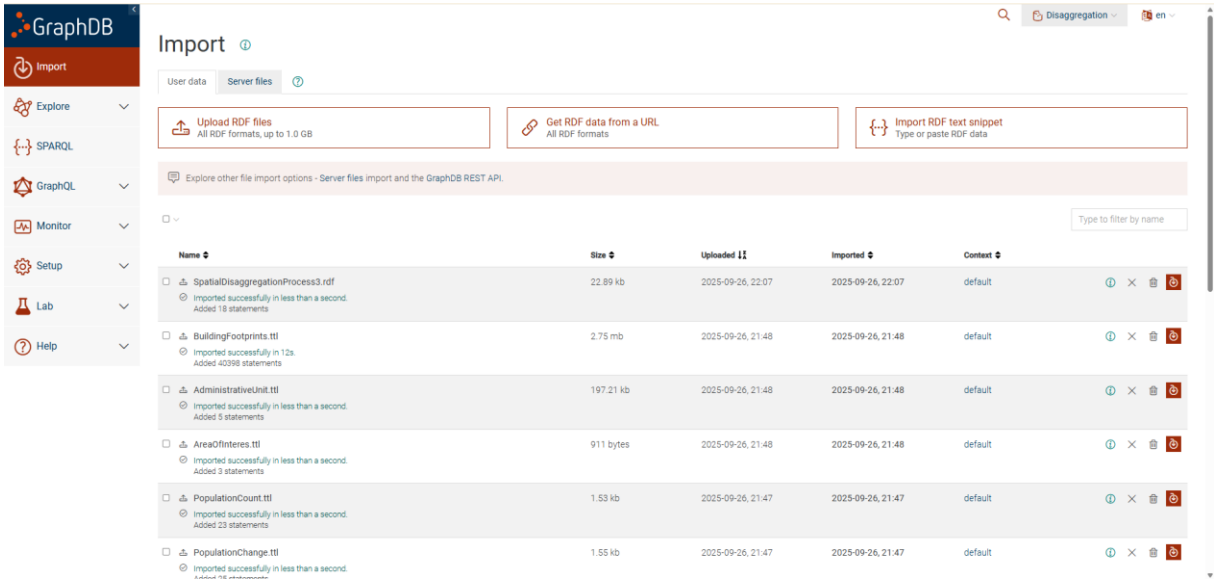


Figure 5.13
GraphDB interface with imported ontologies and schema and data individuals

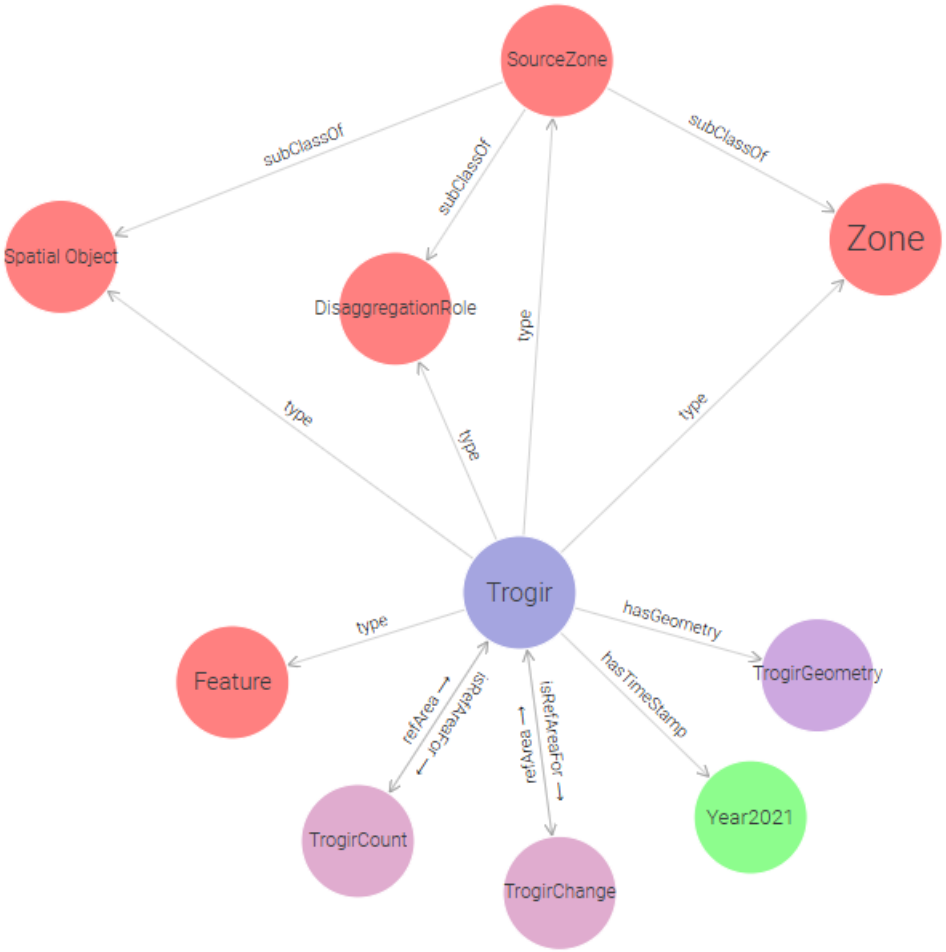


Figure 5.14
Representation of Trogir as source zone of BDASY ontolog in GraphDB database

Execution of the semantical process is done with python orchestrating script. Script uses disaggregation method's execution individual URI as its main input. This URI is used to query for disaggregation method function and its implementation individuals in GraphDB database by sending SPARQL queries to SPARQL Endpoint. Following this pattern, the script reaches algorithm step function individuals that point to algorithm python scripts performing calculations. There is in total seven algorithm python scripts that calculate (1) adjusted population of source zone in time, (2) area ratio, (3) estimated population density, (4) density ratio and (5) total fraction for residential buildings and perform (6) disaggregation and (7) aggregation to area of interest. Each of the algorithm scripts follows the same pattern, it receives URIs as its input, queries the database for data individuals, performs calculations as indicated by the method algorithm, alters the database with new statements and returns URIs of the spatial units for the next function in composition. Scripts are optimised for large amounts of data so querying and inserting new values is done in bulks of 500 objects per shot. While here only general description of Python script is provided, their full code can be found in the appendices section (Appendix D).

5.5 Results

After running the orchestrating process script, results were obtained. Every algorithm script made alterations to the database, so the results were attached to relevant concept URIs and can be extracted from the database using SPARQL queries.

Temporal adjustment disaggregation step calculated adjusted population counts of the Trogir administrative unit using Simple Temporal Adjustment algorithm step. The population of Trogir for year 2022 is calculated by adding population changes of 2022 to population counts of Census 2021. The resulting population count is attached directly to the Trogir source zone using *bdasy:hasAdjustedPopulation* datatype property. The extracted values of population count, change and adjusted population tied to Trogir area are extracted from the database using SPARQL query (Figure 5.15).

Unnamed

```
PREFIX popd: <http://www.example.com/ontology/popdcore#>
PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
PREFIX popcnt: <http://www.example.com/data/popcount/>
PREFIX popcng: <http://www.example.com/data/popchange/>
PREFIX qb: <http://purl.org/linked-data/cube#>

SELECT ?sourceZone ?populationCount ?populationChange ?adjustedPopulation
WHERE {
  ?sourceZone a popd:SourceZone ;
        bdasy:hasAdjustedPopulation ?adjustedPopulation .
  ?trogirCountObservation a qb:Observation ;
        sdmx-dimension:refArea ?sourceZone ;
        popcnt:populationCount ?populationCount .
  ?trogirChangeObservation a qb:Observation ;
        sdmx-dimension:refArea ?sourceZone ;
        popcng:populationChange ?populationChange .
}
```

Run

keyboard shortcuts

	sourceZone	populationCount	populationChange	adjustedPopulation
1	admunit:Trogir	"12393"^^xsd:integer	"-81"^^xsd:integer	"12312.0"^^xsd:double

Figure 5.15
Temporally adjusted population counts: in a) SPARQL query to reach population count, change and adjusted population tied to Trogir source zone URI and in b) resulting values stored in the database

Weight computation step calculated disaggregation weights using area ratio, estimated population density, density ratio and total fraction algorithm steps. Calculated values were attached to residential buildings using *bdasy:hasArea*, *hasAreaRatio*, *hasPopulationDensity*, *hasDensityRatio*, *hasTotalFraction* datatype properties and are extracted from the database using SPARQL query (Figure 5.16).

Unnamed

```
PREFIX sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#>
PREFIX admunit: <http://www.example.com/data/admunit/>
PREFIX popd: <http://www.example.com/ontology/popdcore#>
PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
PREFIX popcnt: <http://www.example.com/data/popcount/>
PREFIX popcng: <http://www.example.com/data/popchange/>
PREFIX qb: <http://purl.org/linked-data/cube#>

SELECT ?residentialBuilding ?area ?areaRatio ?populationDensity ?densityRatio ?
totalFraction
WHERE {
  ?residentialBuilding a bdasy:BuildingFootprint ;
    bdasy:hasArea ?area ;
    bdasy:hasAreaRatio ?areaRatio ;
    bdasy:hasPopulationDensity ?populationDensity ;
    bdasy:hasDensityRatio ?densityRatio ;
    bdasy:hasTotalFraction ?totalFraction.
}
```

Run

keyboard shortcuts

Showing results from 0 to 1000 of 5936. Query took 0.2s, moments ago.

	residentialBuil...	area	areaRatio	populationDen...	densityRatio	totalFraction
1	build:4393693	"0.0633300591768 0486"^^xsd:double	"1.1723900652686 485e-07"^^xsd:double	"0.0227924411744 02597"^^xsd:double	"0.0001684636118 5983747"^^xsd: double	"1.1723900652686 489e-07"^^xsd:double
2	build:12748790	"0.7222322261894 311"^^xsd:double	"1.3370236784990 61e-06"^^xsd:double	"0.0227924411744 02597"^^xsd:double	"0.0001684636118 5983747"^^xsd: double	"1.3370236784990 614e-06"^^xsd:double
3	build:12114993	"48.114500000438 24"^^xsd:double	"8.9071386524998 56e-05"^^xsd:double	"0.0227924411744 02597"^^xsd:double	"0.0001684636118 5983747"^^xsd: double	"8.9071386524998 59e-05"^^xsd:double
4	build:12115435	"16.937999999608 03"^^xsd:double	"3.1356267755287 305e-05"^^xsd:double	"0.0227924411744 02597"^^xsd:double	"0.0001684636118 5983747"^^xsd: double	"3.1356267755287 31e-05"^^xsd:double
5	build:12117027	"24.225550002205 64"^^xsd:double	"4.4847256606791 81e-05"^^xsd:double	"0.0227924411744 02597"^^xsd:double	"0.0001684636118 5983747"^^xsd: double	"4.4847256606791 83e-05"^^xsd:double

Figure 5.16
Disaggregation weights per residential building: in a) SPARQL query to reach building area, area ratio, estimated population density, density ratio and total fraction attached to residential buildings URIs and in b) resulting values stored in the database

Disaggregation step resulting values are calculated from temporally adjusted population and total fraction per residential building. Values are attached to buildings using *bdasy:hasDisaggregatedPopulation* data property and can be reached via building URI (Figure 5.17).

Unnamed

```
PREFIX build: <http://www.example.com/data/building/>
PREFIX sdmx-dimension: <http://purl.org/linked-data/sdmx/2009/dimension#>
PREFIX admunit: <http://www.example.com/data/admunit/>
PREFIX popd: <http://www.example.com/ontology/popdcore#>
PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
PREFIX popcnt: <http://www.example.com/data/popcount/>
PREFIX popcng: <http://www.example.com/data/popchange/>
PREFIX qb: <http://purl.org/linked-data/cube#>

SELECT ?residentialBuilding ?disaggregatedPopulation
WHERE {
  ?residentialBuilding a bdasy:BuildingFootprint ;
    bdasy:hasDisaggregatedPopulation ?disaggregatedPopulation.
}
```

Run

keyboard shortcuts

Showing results from 0 to 1000 of 5936. Query took 0.1s, moments ago.

	residentialBuilding	disaggregatedPopulation
1	build:4393693	"0.0014434466483587605"^^xsd:double
2	build:12748790	"0.016461435529680445"^^xsd:double
3	build:12114993	"1.0966469108957826"^^xsd:double
4	build:12115435	"0.3860583686030974"^^xsd:double
5	build:12117027	"0.552159423342821"^^xsd:double

Figure 5.17
Disaggregated population per residential building URI: in a) SPARQL query to reach disaggregated population values and in b) resulting values stored in the database

In the final step, total population within arbitrary spatial unit (target zone) was calculated as a sum of values per residential buildings. Aggregated value was attached to area of interest individual in the database with *bdasy:hasAggregatedPopulation* data property (Figure 5.18). This value represents final output of the disaggregation in proposed BDASY method. Setting a SPARQL query on area of interest URI this value can be extracted from the database and used externally (Figure 5.18).

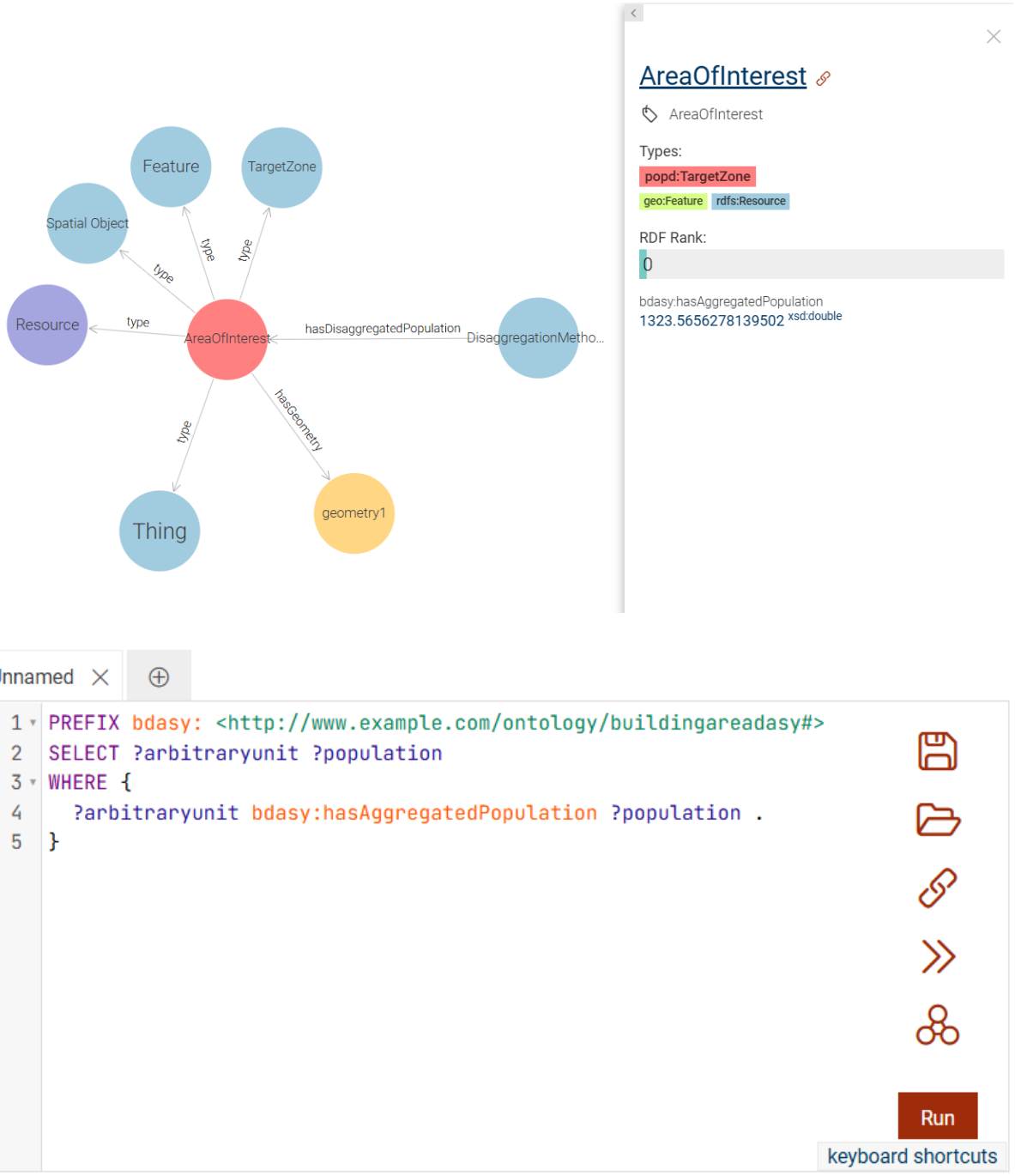


Figure 5.18
Aggregated population of arbitrary target zone: in a) population value within graph database, b) SPARQL query to reach disaggregated population values and in c) resulting values stored in the database

5.6 Discussion

Disaggregated population for Trogir administrative unit proved usability of BDASY ontology in population disaggregation process. Each individual result shown in section 5.5 is directly impacted by semantical descriptions of data flow in the ontology schema. This was enabled with several key design choices of schema individuals supported by BDASY ontology. Firstly, the entire process was described as a composition of compositions which allowed to define only main inputs of the disaggregation method. These values were then passed to each disaggregation step based on semantical linking established by composition individuals. Secondly, main input values were not given as explicit data URIs, but as a SPARQL queries that queried for individuals of interest. This served well within weight computation step in which number of residential buildings was not a priory known and was to be determined by the dataset itself. Further on, algorithm scripts performed calculations based on inputs passed by the process workflow. These inputs were either SPARQL queries or lists of resulting data URIs from previous process step. This setup allowed conceptual description of data flow, where each algorithm step used only descriptions of input and output, query string or list, and not the actual data. This way, entire process remained simplified and not bothered with specific types of output of every algorithm step. Finally, main orchestrating script allowed to combine workflow of BDASY ontology schema with computations in algorithm scripts to create automated process. The script used disaggregation method's execution instance URI to enter ontology schema and read execution sequence from descriptions of the workflow. This allowed fully automated processing where user in only responsible of providing main input data, while the rest of the execution is done based on the knowledge contained in ontology schema.

Discussion

The primary goal of the research presented in this thesis was to develop an ontological model for flexible and automated spatiotemporal disaggregation of population across arbitrary spatial tessellations through the application of Semantic Web technologies. four specific objectives were formulated to ensure that the model is both sustainable and consistent with established disaggregation practices as well as existing knowledge within the Semantic Web.

First objective was to define key concepts and spatial relations in spatial disaggregation methods that should be used as a backbone for the creation of ontological model. Within POPDO ontology, a general disaggregation method was described with several main classes that capture core domain knowledge. These classes consider two different but complementary parts of the disaggregation process, the domain, i.e. data used in the process, and the process itself, i.e. how the data is used to produce the result. Within domain part, proposed SourceZone, TargetZone and AncillaryData classes were introduced as semantical extensions of geospatial data. While this aligns with the consideration that it is the geospatial component of population data that allows spatial disaggregation, these classes within POPDO are considered geospatial roles rather than actual data. Such distinction between real data representation and roles in the disaggregation process allowed geospatial data to be represented in their native domain ontology and play the specific role required by the disaggregation process. This also allowed the distinction of semantic logic in what data really is and what it represents in the process. Process part of ontology with proposed DisaggregationMethod, DisaggregationStep and AlgorithmStep classes allowed modular and flexible description of the disaggregation process. While DisaggregationMethod is a general method that gathers specific procedures in a process, DisaggregationStep and AlgorithmStep provide the basis for multilevel disaggregation descriptions. Such an approach allows grouping of procedures of Algorithm steps within specific step of the disaggregation and thus creation of simple or more complex disaggregation methods. Timestamping of population data was assured with a relation hasTimeStamp to time reference. Linking geospatial and population data of POPDO to their time stamp allows user to choose the most relevant temporal version of the data and thus the creation of the most accurate population

estimation within a chosen disaggregation method. No spatial relations specific to spatial disaggregation were explicitly embedded in the POPDO ontology as literature review revealed these are method specific and used depending on the employed method algorithm. Proposed main classes and property of POPDO align with literature review on shared characteristics of spatiotemporal disaggregation methods. This ensures that the POPDO ontology is top-level ontology, universal enough it can be applied in case of any disaggregation process.

Second objective of the thesis aimed to ensure interoperability of the proposed POPD ontology model by relying on existing knowledge in relevant domain and task ontologies. GeoSPARQL, QB (RDF Data Cube), OWL-Time and FNO (The Function Ontology) proved to have captured domain knowledges relevant for different parts of POPDO. GeoSPARQL formed the geospatial domain ontology in POPDO for several reasons. While it supports description of vector data via its geo:Feature and geo:Geometry classes, it also includes SpatialObject class which is general enough to accommodate other data types. Modern disaggregation methods include different kinds of ancillary data which can be both vector and raster, so SpatialObject class of GeoSPARQL served as a meta class that can accommodate these diversities. This means that in the future, when raster ontologies are developed, they can be easily integrated in POPDO via relation to SpatialObject class. Additionally, GeoSPARQL is an OGC recommended standard, so it is widely used for representation of geospatial data in Semantic Web. Its reuse in POPDO ensured wide ontology applicability and existence of large number of geospatial data directly applicable in the disaggregation process. QB vocabulary in POPDO is used for representation of population data. While POPDO reused only main QB classes and properties, linking of these within full QB vocabulary allows population data to be modelled as part of statistical records and as such reused within disaggregation process. Again, it is a recommendation standard (by W3C) for statistical data description which means all available data published in QB can be directly reused in POPDO. Further on, being inspired by SDMX ontology, QB includes possibility of linking to spatial units so no additional adjustments to the ontology were needed to fit the POPDO requirements. Temporal component of POPDO reused Instant class from OWL-Time ontology. Main

requirement of temporal representation in POPDO was to add time stamps to data to assess its suitability for disaggregation needs and *time:Instant* class provided the semantics needed. While reusing only one class from OWL-Time, its linkage to remaining classes in profile will allow to add disaggregation data in relation to other temporal concepts and give broader temporal context to disaggregation process. OWL-Time is a W3C candidate recommendation so its integration in POPDO ensured standardized time representation of concepts for the disaggregation process. Finally, FNO concepts shaped the process domain of the POPDO ontology. FNO is a light-weight ontology which means its expressivity is optimised for simple application in broad range of use cases. Within POPDO, main advantage of FNO is that it is general enough to describe a process as composition of compositions which fits the idea of disaggregation being modular and flexible. Although FNO ontology has no official recommendation status, its main benefit is that, in contrast to other process ontologies, it is technology independent meaning that disaggregation process can be implemented as a web service using different kinds of technologies which in turn allows wide range of users in wide range of application areas. Existing domain and task ontologies showed maturity of Semantic Web to cover domain knowledge of relevant concepts in POPDO so they could be reused with very few interventions. This makes POPDO fully interoperable with existing knowledge while disaggregation specific concepts add only necessary new knowledge to the knowledge base.

Within the third research objective, disaggregation specific knowledge should have been tied to existing ontologies to develop a model of spatial disaggregation process. While aforementioned domain ontologies describe how data is to be represented in Semantic Web, task ontologies model components of the process that uses data. These different perspectives over real-world entities exposed conflict on semantical interpretation of domain data when compared to roles domain data plays in the disaggregation process. Proposed three-model ontology design for POPDO solved this conflict by grouping knowledge into semantically coherent groups. POPDOd model of POPDO is intended to establish semantical framework for data representation by keeping representation knowledge within a single layer of ontology. By reusing highly conceptual classes for the

creation of POPDOd base (*geo:SpatialObject*, *qb:Observation* and *time:Instant*), knowledge within domains can be easily extended without considering roles in POPDOr. POPDOr builds on top of concepts from POPDOd, i.e. classes from POPDOr serve as intermediator between POPDOp and POPDOd so process inputs and outputs do not reach data individuals directly. Because of this data are not explicitly linked to a disaggregation method but to a role placeholder which allows it to be reused in different methods without alterations. Also, to keep model semantically clean, POPDOr role classes contain restrictions on their members from POPDOd. Source Zone members are geospatial objects linked to population counts, ancillary data are geospatial data different from source zone, and target zone is unit different from the source zone. This ensures clear distinction of roles in the disaggregation process, so the data does not violate logic during disaggregation process. POPDOp model captures the knowledge of task ontology extended with disaggregation specific process classes. These concepts are not semantically aligned with data representation knowledge so POPDO considers them separately. This allows to make distinction between what data is, what is its role and how it is used within the disaggregation process. It also leaves option to make ontology extensions without altering existing semantics. Further on, by making DisaggregationMethod, DisaggregationStep and AlgorithmStep subclasses of *fno:Function* in POPDOp, disaggregation process can reuse FNO functionalities, which means it can be described as composition of compositions. This does not only allow composition directed dataflow and modularity in creation of semantically described disaggregation method, but it also ensures reusability of disaggregation and algorithm steps across methods without making description alterations. Structured in a three-domain approach, POPDO ontology is flexible and modular which were the main requirements in the ontology development process.

Fourth research objective aimed to test the developed disaggregation model on a real case scenario. This aimed to prove three things. First, that POPDO ontology is abstract enough it can fit method-specific concepts without altering ontology structure, second, that ontology model is flexible enough to support any kind of procedural executions and third that it can support semantical description of the workflow and produce results. The testing

was performed using a Building Area Dasymetric Mapping method (BDASY), with its method specific types of geospatial data and algorithm calculation steps. BDASY concepts fit easily with POPDO ontology: census population data fit the statistical descriptions of QB concepts, building footprints extended geospatial classes to form more specific types of geospatial data and algorithm calculation steps were added as Algorithm step functions. This proved POPDO to be highly conceptual as it accommodated method specific concepts by making them more specific types of the core POPDO concepts. Further on, BDASY disaggregation method was proposed to include seven algorithm steps for different phases of the disaggregation process. These were easily added in the POPDO ontology as its multilevel function approach allowed to describe procedures and group them into disaggregation phases. This proved POPDO to be flexible and scalable to meet specific needs of different disaggregation methods. At last, BDASY ontology was implemented as dynamic and real time semantic web service. This required full semantic workflow description and multilevel procedure execution which POPDO successfully supported. As it reuses FNO, execution of BDASY disaggregation combined workflow semantics with procedural execution in Python which proved POPDO ontology to be powerful approach in the population disaggregation domain.

In a broader context, successfully disaggregated data using an arbitrary disaggregation method based on population disaggregation ontology (POPDO) emphasized ontology approach potential in the domain of population disaggregation. When compared to traditional techniques like standalone tools (e.g. Swanwick et al., 2022; Monteiro et al., 2018; Stevens et al., 2015) and platform extensions (e.g. Sleeter and Gould, 2008; Qiu et al., 2012), population disaggregation based on semantical knowledge has several main advantages. First, knowledge of the disaggregation process within POPDO is embedded in logical axioms, so the computer can perform disaggregation without an operator of sufficient knowledge and data manipulation skills. This allows easier access to disaggregated population data for users outside data community. Further on, knowledge stored in POPDO ontology can be reused and build upon across different disaggregation methods, which allows the user to choose between wider pallet of off-the-shelf disaggregation methods. Also, disaggregation via POPDO ontology has no data

preprocessing as data is already within the same data model. Another benefit is that cloud of semantical data is constantly growing so more disaggregation relevant data is available for use. When compared to web service tool solution proposed by Batsaris & Zafeirelli (2023), POPDO based disaggregation shows similarities and some upgrades. In both cases disaggregation is automated as a web service easing it for user to reach disaggregated data, but ontology approach is not limited to only specific data sources and methods and data can be easily extended using new knowledge in ontology. For existing web service tool this is not the case as new approaches and data would still be fixed within hardcode of programming language.

However, POPDO also has limitations when it comes to population disaggregation process. For example, its domain layer mostly reuses data from existing ontologies which may become limiting factor if ontologies are not maintained or become deprecated. Also, model considers statistical region to be aligned with a definition of polygon in geospatial ontology. If this is not the case, statistical units might not be precisely linked to its spatial representation. Additional limitation is that the model considers temporal component as attribute of geospatial and population data. This leaves user of the ontology to reason about which data is suitable for the disaggregation considering its different and often irregular temporal granularities. Further on, role layer of POPDO is method oriented so it keeps clean distinction of roles within the disaggregation process. Potentially, this might be limiting if e.g. target zone is to be used as a source zone of another disaggregation method. Process layer of POPDO considers only procedures of disaggregation method while it neglects modelling uncertainty and error propagation. This might be important if certainty of disaggregation is relevant for further population data usage. Finally, POPDO ontology does not consider semantical description of the algorithm, and it is left to implementation technology to perform calculations based on specific disaggregation method logic. Additional limitation of the current Semantic Web, and POPDO is the lack of ontologies describing raster data. As modern disaggregation methods highly rely on remote sensing imagery and other raster data, this may pose a significant limitation on usability of POPDO ontology. Currently, pixels of raster structure can be considered vectors, i.e. polygons in the ontology, but this requires additional efforts in data

preprocessing. However, POPDO is abstract enough so it considers raster data and can be easily extended once raster geospatial data ontologies are developed.

Based on the proposed ontology model for population disaggregation process and the results obtained, the evaluation of the research hypotheses can be articulated as follows:

1. *Methods of spatial disaggregation for population data can be conceptually modelled as procedures in an ontological model, where procedures, spatial relations, and key components, such as input data, parameters, and outputs — are defined.*

Drawing on the conclusions derived from specific objectives one, two, and three, this hypothesis is confirmed. Population data disaggregation methods can indeed be conceptually represented as procedural constructs, and the overall process can be semantically described as a coherent workflow.

2. *Modelling spatial disaggregation methods as ontological procedures, it is possible to automate the disaggregation of population data.*

On the basis of findings associated with specific objectives three and four, this hypothesis is confirmed. When spatial disaggregation methods are modelled as ontological workflow procedures, they can support the automated execution of disaggregation tasks and facilitate the generation of disaggregated population data.

Conclusion

The research in this thesis presented the development of an ontology-based model for the spatiotemporal disaggregation of population data. To address the limitations of existing approaches and facilitate the production of more accurate spatial representations of population distribution, the Population Disaggregation Ontology (POPDO) is proposed as a formal model for representing the disaggregation process through semantic descriptions of sequential procedures, relevant datasets, and their respective roles. The research demonstrated that integrating domain-specific knowledge with computational logic through ontology engineering can enhance both conceptual clarity and automation capabilities in the field of population disaggregation.

The development of the Population Disaggregation Ontology (POPDO) aimed to establish a flexible ontological framework capable of supporting automated spatiotemporal disaggregation of population data. This process was guided by four specific objectives designed to ensure that the ontology remained both generic and sufficiently expressive for its intended applications. The first objective, "Define key concepts and spatial relations in spatial disaggregation methods," ensured that POPDO encapsulated all domain-relevant concepts in a highly abstract form, enabling its applicability regardless of the specific disaggregation method employed. The second objective, "Ensure interoperability of the proposed ontological model," focused on facilitating the broad usability and adoption of POPDO through the integration of established and widely accepted standards and vocabularies, including GeoSPARQL, OWL-Time, the RDF Data Cube Vocabulary, and the Function Ontology. The third objective, "Develop an ontology model for modelling spatial disaggregation procedures," guided the semantic linking of disaggregation-relevant concepts into a coherent ontological structure. The three-layer architecture of POPDO, comprising POPDOd, POPDOr, and POPDOp, enabled the harmonization of differing semantic interpretations into a unified and operational ontology. Finally, the fourth objective, "Test the developed disaggregation model," facilitated an evaluation of the ontology's practical applicability in a real-world context. Empirical testing conducted using data from the administrative unit of Trogir confirmed both the conceptual soundness and operational feasibility of the POPDO-based disaggregation approach. The results further highlighted the ontology-driven methodology's advantages in terms of

transparency, reusability, interoperability, and automation potential, demonstrating its viability as a robust alternative to traditional disaggregation techniques.

The identified limitations of the Population Disaggregation Ontology (POPDO) provide valuable guidance for future research and development efforts, particularly in enhancing disaggregation accuracy and advancing raster data ontologies, which are recognized as priority research domains. Addressing these aspects will ultimately ensure that POPDO fully supports advanced population disaggregation methodologies while incorporating mechanisms for assessing and maintaining result accuracy. Despite its current limitations, the proposed POPDO model, validated through empirical testing, offers notable contributions to both theoretical understanding and practical implementation. From a theoretical standpoint, POPDO represents the first semantic web–based approach to conceptualizing population disaggregation as a process. Its conceptual structure and integration with existing Semantic Web ontologies expand the knowledge base and promote further development of computer-automated disaggregation systems. Practically, POPDO establishes a foundation for developing automated web services capable of delivering disaggregated population data to a broad spectrum of users, thereby enhancing data accessibility and supporting the advancement of applications that depend on accurate population information.

Within the broader paradigm of contemporary data sharing and utilization, the Semantic Web and its associated applications are assuming increasingly significant roles. National mapping agencies worldwide, such as Ordnance Survey and the United States Geological Survey (USGS), recognize semantic technologies as a transformative advancement that facilitates the dissemination and utilization of geospatial data. These organizations anticipate a growing demand for geospatial datasets that are compatible with Semantic Web services and are actively working to address this emerging need. This trend has direct implications for the adoption of semantically enabled applications required to fully leverage the potential of such data. In this context, application ontologies such as the Population Disaggregation Ontology (POPDO) will play a crucial role.

REFERENCES

Al-Feel, H. T., Koutb, M., & Suoror, H. (2008). Semantic Web on Scope: A New Architectural Model for the Semantic Web. *Journal of Computer Science*, *4*(7), 613–624. https://doi.org/10.3844/jcssp.2008.613.624

Allemang, D., & Hendler, J. (2011). *Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL* (Second). Morgan Kaufmann.

Armstrong, M. (1988). Temporality in spatial database. *Proceedings: Acccessing the World*, *December*.

Arp, R., Smith, B., & Spear, A. D. (2015). *Building Ontologies with Basic Formal Ontology*. MIT Press.

Baškarada, S., & Koronios, A. (2013). Data, information, knowledge, wisdom (DIKW): A semiotic theoretical and empirical exploration of the hierarchy and its quality dimension. *Australasian Journal of Information Systems*, *18*(1), 5–24. https://doi.org/10.3127/ajis.v18i1.748

Batsaris, M., & Zafeirelli, S. (2023). PoD : A Web Tool for Population Downscaling Using Areal Interpolation and Volunteered Geographic Information. *European Journal of Geography*, *14*(4), 22–36. https://doi.org/10.48088/ejg.m.bat.14.4.022.036

Baučić, M. (2014). *Geoprostorne semantičke mreže u upravljanju izvanrednim situacijama u zračnim lukama*. University of Zagreb Faculty of Geodesy.

Baynes, J., Neale, A., & Hultgren, T. (2022). Improving intelligent dasymetric mapping population density estimates at 30 m resolution for the conterminous United States by excluding uninhabited areas. *Earth System Science Data*, *14*, 2833–2849. https://doi.org/10.5194/essd-14-2833-2022

Bednár, P., Ivančáková, J., & Sarnovský, M. (2024). Semantic Composition of Data Analytical Processes. *Acta Polytechnica Hungarica*, *21*(2), 167–185. https://doi.org/10.12700/APH.21.2.2024.2.9

Beller, A., Giblin, T., Le, K., Litz, S., Kitter, T., & Schimel, D. (1991). Temporal gis prototype for global change research. *GIS/LIS 1991 Proceedings, Volume 2*, 752–765.

Belozerov, A. A., & Klimov, V. V. (2022). ScienceDirect ScienceDirect Semantic Web Technologies : Issues and Possible Ways of Development. *Procedia Computer Science*, *213*, 617–622. https://doi.org/10.1016/j.procs.2022.11.112

Bernard, C., Villanova-oliver, M., & Gensel, J. (2022). Theseus : A framework for managing knowledge graphs about geographical divisions and their evolution. *Transactions in GIS*, *26*, 3202–3224. https://doi.org/10.1111/tgis.12988

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web: A New Form of Web Content that is Meaningful to Computers will Unleash a Revolution of New Possibilities. *Scientific American*, *284*, 34–43. https://doi.org/10.1145/3591366.3591376

Bielecka, E. (2005). A dasymetric population density map of Poland. *Proceedings of the 22nd International Cartographic Conference*, 9–15.

Borgida, A., & Brachman, R. J. (2010). Conceptual Modeling with Description Logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, & P. F. Patel-Schneider (Eds.), *The Description Logic Handbook: Theory, Implementation and Applications* (Second, pp. 377–401). Cambridge University Press.

Borgo, S., Ferrario, R., Gangemi, A., Guarino, N., Masolo, C., Porello, D., Sanfilippo, E. M., & Vieu, L. (2022). DOLCE : A Descriptive Ontology for Linguistic and Cognitive Engineering 1. *Applied Ontology*, *17*(1), 45–69. https://doi.org/https://doi.org/10.3233/AO-210259

Briant, A., Combes, P. P., & Lafourcade, M. (2010). Dots to boxes: Do the size and shape of spatial units jeopardize economic geography estimations? *Journal of Urban Economics*, *67*(3), 287–302. https://doi.org/10.1016/j.jue.2009.09.014

Calka, B., Bielecka, E., & Zdunkiewicz, K. (2016). Redistribution population data across a regular spatial grid according to buildings characteristics. *Geodesy and*

Cartography, *65*(2), 149–162. https://doi.org/10.1515/geocart-2016-0011

Calka, B., Nowak, J., Costa, D., Bielecka, E., Nowak, J., & Costa, D. (2017). Fine scale population density data and its application in risk assessment. *Geomatics, Natural Hazards and Risk*, *8*(2), 1440–1455. https://doi.org/10.1080/19475705.2017.1345792

Cartagena-Colón, M., Mattei, H., & Wang, C. (2022). Dasymetric Mapping of Population Using Land Cover Data in JBNERR, Puerto Rico during 1990–2010. *Land*, *11*. https://doi.org/10.3390/land11122301

Ch, R., Martin, D. A., & Vargas, J. F. (2021). Measuring the Size and Growth of Cities Using Nighttime. *Journal of Urban Economics*, *125*. https://doi.org/https://doi.org/10.1016/j.jue.2020.103254

Cheng, Z., Wang, J., & Ge, Y. (2022). Mapping monthly population distribution and variation at 1-km resolution across China. *International Journal of Geographical Information Science*, *36*(6), 1166–1184. https://doi.org/10.1080/13658816.2020.1854767

Christen, P., & Schnell, R. (2023). Thirty-three myths and misconceptions about population data: from data capture and processing to linkage. *International Journal of Population Data Science*, *8*(1). https://doi.org/10.23889/ijpds.v8i1.2115

Comber, A., & Zeng, W. (2019). Spatial interpolation using areal features: A review of methods and opportunities using new forms of data with coded illustrations. *Geography Compass*, *13*. https://doi.org/10.1111/gec3.12465

Comber, A., & Zeng, W. (2022). *Areeal Interpolation*. The Geographic Information Science & Technology Body of Knowledge (2nd Quarter 2022 Edition). https://doi.org/10.22224/gistbok/2022.2.2

Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2003). Methodologies, tools and languages for building ontologies. Where is their meeting point? *Data and Knowledge Engineering*, *46*(1), 41–64. https://doi.org/10.1016/S0169-023X(02)00195-7

Cova, T. J. (2017). Data Model, F -Objects and O -Fields. In D. Richardson, N. Castree, M. F. Goodchild, A. Kobayashi, W. Liu, & R. A. Marston (Eds.), *International Encyclopedia of Geography*. https://doi.org/10.1002/9781118786352.wbieg0481

Cova, T. J., & Goodchild, M. F. (2002). Extending geographical representation to include fields of spatial objects. *International Journal of Geographical Information Science*, *16*(6), 509–532. https://doi.org/10.1080/13658810210137040

Croatian Bureau of Statistics. (2023). *Natural Change in Population, 2022*. https://podaci.dzs.hr/media/yh2da1ot/si-1718_prirodno-kretanje-stanovnistva-u-2022.pdf

Cromley, R. G., Hanink, D. M., & Bentley, G. C. (2011). A Quantile Regression Approach to Areal Interpolation. *Annals of the Association of American Geographers*, *102*(4), 763–777. https://doi.org/10.1080/00045608.2011.627054

Cyganiak, R., Field, S., Gregory, A., Halb, W., & Tennison, J. (2010). Semantic statistics: Bringing together SDMX and SCOVO. In C. Bizer, T. Heath, T. Berners-Lee, & M. Hausenblas (Eds.), *Proceedings of the WWW2010 Workshop on Linked Data on the Web (LDOW2010)* (Vol. 628). CEUR Workshop Proceedings. online CEUR-WS.org/Vol-628/

De Meester, B., Dimou, A., & Kleedorfer, F. (2023). *The Function Ontology*. Retrieved 24 September 2025. https://fno.io/spec/

De Meester, B., Dimou, A., Verborgh, R., & Mannens, E. (2016). An Ontology to Semantically Declare and Describe Functions. In H. Sack, G. Rizzo, N. Steinmetz, D. Mladenić, S. Auer, & C. Lange (Eds.), *The Semantic Web* (pp. 46–49). Springer Cham. https://doi.org/https://doi.org/10.1007/978-3-319-47602-5_10

De Meester, B., Seymoens, T., Dimou, A., & Verborgh, R. (2020). Implementation-independent function reuse. *Future Generation Computer Systems*, *110*, 946–959. https://doi.org/https://doi.org/10.1016/j.future.2019.10.006

De Souza Neto, J. B., Moreira, A. M., & Musicante, M. A. (2018). Semantic Web

Services testing: A Systematic Mapping study. *Computer Science Review*, *28*, 140–156. https://doi.org/10.1016/j.cosrev.2018.03.002

Docker. (2025). *Docker context*. Retrieved 21 September 2025. https://docs.docker.com/engine/manage-resources/contexts/

Dreyer, A., Cotton, F., & Duffès, G. (2016). An OWL Ontology for the Common Statistical Production Architecture. *Proceedings of the 4th International Workshop on Semantic Statistics Co-Located with 15th International Semantic Web Conference (ISWC 2016)*, 1–12. http://ceur-ws.org/Vol-1654/article-06.pdf

Eicher, C. L., & Brewer, C. A. (2001). Dasymetric mapping and areal interpolation: Implementation and evaluation. *Cartography and Geographic Information Science*, *28*(2), 125–138. https://doi.org/10.1559/152304001782173727

Espey, J. M., Tatem, A. J., & Thomson, D. R. (2025). Disappearing people: A global demographic data crisis threatens public policy. *Science*, *388*, 1277–1280. https://doi.org/10.1126/science.adx8683

European Commission. (2023). COMMISSION IMPLEMENTING REGULATION (EU) 2023/138 of 21 December 2022 laying down a list of specific high-value datasets and the arrangements for their publication and re-use. *Official Journal of the European Union*, *L 19*, 43–75. https://eur-lex.europa.eu/eli/reg_impl/2023/138/oj/eng

European Parliament. (2019). Directive (EU) on open data and the re-use of public sector information, 2019/1024. *Official Journal of the European Union*, *L 172*, 56–83. http://data.europa.eu/eli/dir/2019/1024/oj

Fecht, D., Cockings, S., Hodgson, S., Piel, B., Martin, D., & Waller, L. A. (2020). Advances in mapping population and demographic characteristics at small-area levels. *International Journal of Epidemiolology*, *49*, 15–25. https://doi.org/10.1093/ije/dyz179

Fendrich, A. N., Neto, E. S. H., Moreira, L. E. M. e., & Neto, D. D. (2022). A scalable method for the estimation of spatial disaggregation models. *Computers and*

Geosciences, *166*. https://doi.org/10.1016/j.cageo.2022.105161

Fina, S., Gerten, C., Pondi, B., Arcy, L. D., Reilly, N. O., Sousa, D., Pereira, M., & Zilio, S. (2022). OS-WALK-EU : An open-source tool to assess health-promoting residential walkability of European city structures. *Journal of Transport & Health*, *27*. https://doi.org/10.1016/j.jth.2022.101486

Flasse, C., Grippa, T., & Fennia, S. (2021). A TOOL FOR MACHINE LEARNING BASED DASYMETRIC MAPPING APPROACHES IN GRASS GIS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLVI-4/W2-*, 55–62. https://doi.org/10.5194/isprs-archives-XLVI-4-W2-2021-55-2021

Frontiersi. (2018). *Delivering Better Automated Systems with the Semantic Web*.

Galvani, A. P., Bauch, C. T., Anand, M., Singer, B. H., & Levin, S. A. (2016). Human – environment interactions in population and ecosystem health. *Proceedings of the National Academy of Sciences*, *113*(51), 14502–14506. https://doi.org/10.1073/pnas.1618138113

GeoNames. (2025). *GeoNames Ontology*. Retrieved 22 September 2025. https://www.geonames.org/ontology/documentation.html

Giacomo, G. De, & Lenzerini, M. (1996). TBox and ABox Reasoning in Expressive Description Logics. In L. Padgham, E. Franconi, M. Gehrke, D. L. McGuinness, & P. F. Patel-Schneider (Eds.), *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR'96)* (pp. 37–48). AAAI Press. https://www.researchgate.net/publication/220957243_TBox_and_ABox_Reasoning_in_Expressive_Description_Logics

Goodchild, M. F. (1992). Geographical Data Modeling. *Computers and Geosciences*, *18*(4), 401–408. https://doi.org/10.1016/0098-3004(92)90069-4

Goodchild, M. F. (2013). Prospects for a Space-Time GIS: Space-Time Integration in Geography and GIScience. *Annals of the Association of American Geographers*,

103(5), 1072–1077. https://doi.org/10.1080/00045608.2013.792175

Goplerud, M. (2016). Crossing the boundaries: An implementation of two methods for projecting data across boundary changes. *Political Analysis*, *24*, 121–129. https://doi.org/10.1093/pan/mpv029

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199–220. https://doi.org/10.1006/knac.1993.1008

Guarino, N. (1998). Formal Ontology adn Information Systems. *Proceedings of the 1st International Conference*, *June*. https://doi.org/10.1007/978-3-642-00834-4_23

Hallot, E., Okende, A., Grippa, T., & Beaumont, B. (2021). A Random Forest Dasymetric Approach For Mapping The Population Distribution At High Spatial Resolution. *EARSeL Joint Workshop - Earth Observation for Sustainable Cities and Communities*.

Han, B., & Howe, B. (2024). SARN: Structurally-Aware Recurrent Network for Spatio-Temporal Disaggregation. *32nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL 2024*, 338–349. https://doi.org/10.1145/3678717.3691295

Hasani, S., Sadeghi-Niaraki, A., & Jelokhani-Niaraki, M. (2015). Spatial Data Integration using Ontology-Based Approach. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XL-1/W5*, 23–25. https://doi.org/10.5194/isprsarchives-XL-1-W5-293-2015

Hedefalk, F., Harrie, L., & Svensson, P. (2014). Extending the Intermediate Data Structure (IDS) for longitudinal historical databases to include geographic data. *Historical Life Course Studies*, *1*, 27–46. https://doi.org/https://doi.org/10.51964/hlcs9289

Hemalatha, M., Uma, V., & Aghila, G. (2012). Time ontology with Reference Event based Temporal Relations (RETR). *International Journal of Web & Semantic Technology*, *3*(1), 23–31. https://doi.org/10.5121/ijwest.2012.3103

Hobbs, J. R., & Pan, F. (2004). An ontology of time for the semantic web. *ACM*

Transactions on Asian Language Information Processing, *3*(1), 66–85. https://doi.org/10.1145/1017068.1017073

Hogan, A. (2020). Web of Data. In *The Web of Data* (1st ed.). Springer Cham. https://doi.org/https://doi.org/10.1007/978-3-030-51580-5

Jacquez, G., Maruca, S., & Fortin, M.-J. (2000). From fields to objects: A review of geographic boundary analysis. *Journal of Geographical Systems*, *2*(3), 221–241. https://doi.org/10.1007/PL00011456

Jama, T., Tenkanen, H., Henrik, L., & Joutsiniemi, A. (2025). Compact city and urban planning : Correlation between density and local amenities. *Environment and Planning B: Urban Analytics and City Science*, *52*(1), 44–58. https://doi.org/10.1177/23998083241250264

Jurisica, I., Mylopoulos, J., & Yu, E. (2004). Ontologies for Knowledge Management : An Information Systems Perspective. *Knowledge Information Systems*, *6*, 380–401. https://doi.org/https://doi.org/10.1007/s10115-003-0135-4

Kaladzavi, G., Traore, Y., & Abba Ari, A. A. (2017). Geo-spatial Domain Ontology: The Case of the Socio-Cultural Infrastructures. *International Journal of Computer Applications*, *178*(1), 42–48. https://doi.org/10.5120/ijca2017915720

Karunarathne, A., & Lee, G. (2019). Estimating hilly areas population using a dasymetric mapping approach: A case of Sri Lanka's highest mountain range. *ISPRS International Journal of Geo-Information*, *8*. https://doi.org/10.3390/ijgi8040166

Katsumi, M. (2025). *Geometry Ontology*. Retrieved 22 September 2025. https://enterpriseintegrationlab.github.io/icity/Geom/doc/index-en.html#http://geovocab.org/geometry

Kević, K., & Kuveždić Divjak, A. (2025). Open Technologies Supporting Linked Open Data Publishing: Croatian Population Census Case Study. In M. Minghini, M. Ciolli, L. Šerić, & T. Hlupić (Eds.), *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* (Vol. 48,

Issues 4/W13-2025, pp. 157–164). https://doi.org/10.5194/isprs-Archives-XLVIII-4-W13-2025-157-2025

Kharbat, G., & El-Ghalayini, H. (2008). Building Ontology from Knowledge Base Systems. In E. Giannapoulou (Ed.), *Data Mining in Medical and Biological Research*. InTech. https://doi.org/doi: 10.5772/6407

King, R. (2019). *An ontology-based Modelling framework for detailed spatio-temporal population estimation* [University of Southampton]. http://eprints.soton.ac.uk/id/eprint/431111

Kjenstad, K. (2006). On the integration of object-based models and field-based models in GIS. *International Journal of Geographical Information Science*, *20*(5), 491–509. https://doi.org/10.1080/13658810600607329

Krisnadhi, A., & Hitzler, P. (2014). Description Logics. In *Encyclopedia of Social Network Analysis and Mining* (pp. 1–6). https://doi.org/10.1007/3-540-44613-3_2

Lam, N. S. N. (1983). Spatial interpolation methods: A review. *The American Cartographer*, *10*(2), 129–150. https://doi.org/10.1559/152304083783914958

Langran, G., & Chrisman, N. R. (1988). A Framework for Temporal Geographic Information. *Cartographica: The International Journal for Geographic Information and Geovisualization*, *25*, 1–14. https://doi.org/https://doi.org/10.3138/K877-7273-2238-5Q6V

Leyk, S., Gaughan, A. E., Adamo, S. B., De Sherbinin, A., Balk, D., Freire, S., Rose, A., Stevens, F. R., Blankespoor, B., Frye, C., Comenetz, J., Sorichetta, A., Macmanus, K., Pistolesi, L., Levy, M., Tatem, A. J., & Pesaresi, M. (2019). The spatial allocation of population: a review of large-scale gridded population data products and their fitness for use. *Earth System Science Data*, *11*, 1385–1409. https://doi.org/10.5194/essd-11-1385-2019

Li, F., Du, J., He, Y., Song, H. Y., Madkour, M., Rao, G., Xiang, Y., Luo, Y., Chen, H. W., Liu, S., Wang, L., Liu, H., Xu, H., & Tao, C. (2020). Time event ontology

(TEO): To support semantic representation and reasoning of complex temporal relations of clinical events. *Journal of the American Medical Informatics Association*, *27*(7), 1046–1056. https://doi.org/10.1093/jamia/ocaa058

Li, X. (2022). Suitability evaluation method of urban and rural spatial planning based on artificial intelligence. *Journal of Intelligent Systems*, *31*(1), 245–259. https://doi.org/https://doi.org/10.1515/jisys-2022-0010

Lin, Y., & Xiao, N. (2024). Exploring the Tradeoff Between Privacy and Utility of Complete-count Census Data Using a Multiobjective Optimization Approach. *Geographical Analysis*, *56*(3), 427–450. https://doi.org/10.1111/gean.12388

Liu, C., Chen, Y., Wei, Y., & Chen, F. (2023). Spatial Population Distribution Data Disaggregation Based on SDGSAT-1 Nighttime Light and Land Use Data Using Guilin, China, as an Example. *Remote Sensing*, *15*. https://doi.org/10.3390/rs15112926

Lloyd, C. T., Chamberlain, H., Kerr, D., Yetman, G., Pistolesi, L., Stevens, F. R., Gaughan, A. E., Nieves, J. J., Hornby, G., MacManus, K., Sinha, P., Bondarenko, M., Sorichetta, A., & Tatem, A. J. (2019). Global spatio-temporally harmonised datasets for producing high-resolution gridded population distribution datasets. *Big Earth Data*, *3*(2), 108–139. https://doi.org/10.1080/20964471.2019.1625151

Maedche, A. (2002). Ontology - Definition & Overview. In *Ontology Learning for the Semantic Web* (pp. 11–27). Springer. https://doi.org/https://doi.org/10.1007/978-1-4615-0925-7_2

Matthews, S. A., & Parker, D. M. (2013). Progress in Spatial Demography. *Demographic Research*, *28*, 271–312. https://doi.org/10.4054/demres.2013.28.10

Mennis, J. (2015). Increasing the Accuracy of Urban Population Analysis with Dasymetric Mapping. *Cityscape*, *17*(1), 115–126. https://doi.org/https://doi.org/10.1559/152304010792194985

Mennis, Jeremy. (2003). Generating Surface Models of Population Using Dasymetric Mapping. *The Professional Geographer*, *55*(1), 31–42.

https://doi.org/10.1111/0033-0124.10042

Mennis, Jeremy, & Hultgren, T. (2006a). Intelligent dasymetric mapping and its application to areal interpolation. *Cartography and Geographic Information Science*, *33*(3), 179–194. https://doi.org/10.1559/152304006779077309

Mennis, Jeremy, & Hultgren, T. (2006b). 'Intelligent'Dasymetric Mapping and Its Comparison to Other Areal Interpolation Techniques. *Proceedings 16th International Research Symposium on Computer-Based Cartography*, 1–9. http://www.cartogis.org/publications/autocarto-2006/mennishultgren.pdf

Mennis, Jeremy, & Hultgren, T. (2005). Dasymetric Mapping for Disaggregating Coarse Resolution Population Data. *Proceedings of the 22nd Annual International Cartographic Conference*, 9–16.

Miller, H. J. (2003). What about people in geographic information science? *Computers, Environment and Urban Systems*, *27*, 447–453. https://doi.org/10.1016/s0198-9715(03)00059-0

Miller, H. J. (2007). Place-Based versus People-Based Geographic Information Science. *Geography Compass*, *1*(3), 503–535. https://doi.org/10.1111/j.1749-8198.2007.00025.x

Mohamad, U. H., Ahmad, M. N., Benferdia, Y., Shapi'i, A., & Bajuri, M. Y. (2021). An Overview of Ontologies in Virtual Reality-Based Training for Healthcare Domain. *Frontiers in Medicine*, *8*, 1–13. https://doi.org/10.3389/fmed.2021.698855

Monteiro, J., Martins, B., Costa, M., & Pires, J. M. (2021). Geospatial data disaggregation through self-trained encoder–decoder convolutional models. *ISPRS International Journal of Geo-Information*, *10*, 1–28. https://doi.org/10.3390/ijgi10090619

Monteiro, J., Martins, B., Murrieta-Flores, P., & Pires, J. M. (2019). Spatial Disaggregation of Historical Census Data Leveraging Multiple Sources of Ancillary Information. *ISPRS International Journal of Geo-Information*, *8*(8). https://doi.org/10.3390/ijgi8080327

Monteiro, J., Martins, B., & Pires, J. M. (2018). A hybrid approach for the spatial disaggregation of socio-economic indicators. *International Journal of Data Science and Analytics*, *5*(2), 189–211. https://doi.org/10.1007/s41060-017-0080-z

Moreira Ribeiro, C. G. (2017). *Scalable and Memory-Efficient Approaches for Spatial Data Downscaling Leveraging Machine Learning* [Tecnico Lisboa]. https://www.dpss.inesc-id.pt/~romanop/files/students/CR/CR-thesis.pdf

Muscetti, M., Rinaldi, A. M., Russo, C., & Tommasino, C. (2022). Multimedia ontology population through semantic analysis and hierarchical deep features extraction techniques. *Knowledge and Information Systems*, *64*, 1283–1303. https://doi.org/10.1007/s10115-022-01669-6

Musen, M. A. (2015). The Protégé Project: A Look Back and a Look Forward. *AI Matters*, *1*(4), 4–12. https://doi.org/https://doi.org/10.1145/2757001.2757003

Nardi, D., & Brachman, R. J. (2010). *An Introduction to Description Logics* (F. Baader, D. Calvanese, D. McGuinness, D. Nardi, & P. F. Patel-Schneider (eds.); Second, pp. 1–43). Cambridge University Press.

National Academy of Sciences Engineering and Medicine. (2022). Ontologies in the behavioral sciences: Accelerating research and the spread of knowledge. In *A Consensus Study Report of The National Academies of Seinces, Engineering and Medicine*. https://doi.org/https://doi.org/10.17226/26464

National Research Council. (2007). *Using the American Community Survey: Benefits and Challenges* (C. F. Citro & G. Kalton (eds.)). The National Academies Press. https://doi.org/10.17226/11901

Neal, I., Seth, S., Watmough, G., & Diallo, M. S. (2022). Census-independent population estimation using representation learning. *Scientific Reports*, *12*. https://doi.org/10.1038/s41598-022-08935-1

Nebiler, M., & Neupert, R. (2020). *Current Census Methodologies*. https://documents.worldbank.org/en/publication/documents-reports/documentdetail/099640011012229715/P173497093d9ac0f0b3e50bea

b3013032c

Netrdová, P., Nosek, V., & Hurbánek, P. (2020). Using areal interpolation to deal with differing regional structures in international research. *ISPRS International Journal of Geo-Information*, *9*(2). https://doi.org/10.3390/ijgi9020126

Nettleton, David. (2014). Incorporating Various Sources of Data and Information. In D Nettleton (Ed.), *Commercial Data Mining: Processing, Analysis and Modeling for Predictive Analytics Projects* (pp. 17–47). Morgan Kaufmann. https://doi.org/https://doi.org/10.1016/B978-0-12-416602-8.00003-0

Neuhaus, F. (2018). What Is an Ontology? *Building Ontologies With Basic Formal Ontology*, 1–26. https://doi.org/10.7551/mitpress/9780262527811.003.0001

Niles, I., & Pease, A. (2001). Towards a Standard Upper Ontology. *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, 2–9. https://doi.org/10.1145/505168.505170

Noy, N. F., & McGuinness, D. L. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology*. https://course.ccs.neu.edu/cs5100f11/resources/noy01.pdf

Official Gazette of The Republic of Croatia. (2020). *Regulation on the Register of Spatial Units*. Retrieved 24 September 2025. https://narodne-novine.nn.hr/clanci/sluzbeni/2020_03_37_808.html

Official Gazette of The Republic of Croatia. (2021). *The Act on the Census of Population, Households and Dwellings in The Republic of Croatia in 2021*. https://dzs.gov.hr/UserDocsImages/Popis 2021/PDF/The Act on the Census.pdf

OGC. (2012). *A Geographic Query Language for RDF Data v1.0*. Retrieved 22 September 2025. https://www.ogc.org/standards/geosparql/

OGC. (2024). *A Geographic Query Language for RDF Data v1.1*. Retrieved 22 September 2025. http://www.opengis.net/doc/IS/geosparql/1.1

Oliveira, A., Fachada, N., & Matos-Carvalho, J. P. (2024). Data Science for Geographic Information Systems. *2024 8th International Young Engineers Forum on*

Electrical and Computer Engineering (YEF-ECE), 1–7. https://doi.org/10.1109/YEF-ECE62614.2024.10624902

Ontotext. (2022). *Ontotext Refine*. Retrieved 25 September 2025. https://www.ontotext.com/products/ontotext-refine/

Ontotext. (2025). *SPARQL Compliance*. Retrieved 21 September 2025. https://graphdb.ontotext.com/documentation/11.1/sparql-compliance.html

Pajares, E., Nieto, R. M., Meng, L., & Wulfhorst, G. (2021). Population disaggregation on the building level based on outdated census data. *ISPRS International Journal of Geo-Information*, *10*(10). https://doi.org/10.3390/ijgi10100662

Peuquet, D. J., & Duan, N. (1995). An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data. *International Journal of Geographical Information Systems*, *9*(1), 7–24. https://doi.org/10.1080/02693799508902022

Peuquet, D., Smith, B., & Brogaard, B. (1998). *The Ontology of Fields, Final Report*.

Polenghi, A., Roda, I., Macchi, M., Pozzetti, A., & Panetto, H. (2022). Knowledge reuse for ontology modelling in Maintenance and Industrial Asset Management. *Journal of Industrial Information Integration*, *27*. https://doi.org/10.1016/j.jii.2021.100298

Pred, A. (1977). The Choreography of Existence : Comments on Hägerstrand ' s Time-Geography and Its Usefulness. *Economic Geography*, *53*(2), 207–221. https://doi.org/https://doi.org/10.2307/142726

Qiu, F., Zhang, C., & Zhou, Y. (2012). The development of an areal interpolation ArcGIS extension and a comparative study. *GIScience and Remote Sensing*, *49*(5), 644–663. https://doi.org/10.2747/1548-1603.49.5.644

Qiu, Y., Zhao, X., Fan, D., Li, S., & Zhao, Y. (2022). Disaggregating population data for assessing progress of SDGs: methods and applications. *International Journal of Digital Earth*, *15*(1), 2–29. https://doi.org/10.1080/17538947.2021.2013553

Raskin, R. G., & Pan, M. J. (2005). Knowlege representation in the semantic web for Earth and environmental terminology (SWEET). *Computers and Geosciences*,

31(9), 1119–1125. https://doi.org/https://doi.org/10.1016/j.cageo.2004.12.004

Raymer, J., Willekens, F., & Rogers, A. (2019). Spatial demography: A unifying core and agenda for further research. *Population, Space and Place*, *25*(4). https://doi.org/10.1002/psp.2179

Reiter, D., Jehling, M., & Hecht, R. (2023). Benefits of using address-based dasymetric mapping in micro-level census disaggregation. *AGILE: GIScience Series*, *4*, 1–7. https://doi.org/10.5194/agile-giss-4-38-2023

Renner, K., Schneiderbauer, S., Pruß, F., Kofler, C., Martin, D., & Cockings, S. (2018). Spatio-temporal population modelling as improved exposure information for risk assessments tested in the Autonomous Province of Bolzano. *International Journal of Disaster Risk Reduction*, *27*, 470–479. https://doi.org/10.1016/j.ijdrr.2017.11.011

Sapena, M., Kühnl, M., Wurm, M., Patino, J. E., Duque, J. C., & Taubenböck, H. (2022). Empiric recommendations for population disaggregation under different data scenarios. *PLoS ONE*, *17*(9). https://doi.org/10.1371/journal.pone.0274504

Shili, M., & Sohaib, O. (2025). Geographic recommender systems in e-commerce based on population. *PeerJ Computer Science*, *11:e2525*, 1–26. https://doi.org/10.7717/peerj-cs.2525

Siabato, W., Claramunt, C., Ilarri, S., & Manso-Callejo, M. A. (2019). A survey of modelling trends in temporal GIS. *ACM Computing Surveys*, *51*(2), 41. https://doi.org/10.1145/3141772

Silva, E., Guerrero, V. M., & Peña, D. (2011). Temporal disaggregation and restricted forecasting of multiple population time series. *Journal of Applied Statistics*, *38*(4), 799–815. https://doi.org/10.1080/02664761003692316

Sleeter, R., & Gould, M. (2008). Techniques and Methods 11–C2. In *Geographic Information System Software to Remodel Population Data Using Dasymetric Mapping Methods*.

Smith, B., & Mark, D. M. (1998). Ontology and Geographic Kinds. In T. Poiker & N.

Chrisman (Eds.), *Proceedings. 8th International Symposium on Spatial Data Handling (SDH'98)* (pp. 308–320).

Song, J., Tong, X., Wang, L., Zhao, C., & Prishchepov, A. V. (2019). Monitoring finer-scale population density in urban functional zones: A remote sensing data fusion approach. *Landscape and Urban Planning*, *190*. https://doi.org/10.1016/j.landurbplan.2019.05.011

State Geodetic Administration. (2025). *SGA Geoportal*. Retrieved 24 September 2025. https://geoportal.dgu.hr/

STATO Project. (2014). *Statistics Ontology STATO*. Retrieved 23 September 2025. https://stato-ontology.org/index.jsp

Stevens, F. R., Gaughan, A. E., Linard, C., & Tatem, A. J. (2015). Disaggregating Census Data for Population Mapping Using Random Forests with Remotely-Sensed and Ancillary Data. *PLoS ONE*, *10*(2), 1–22. https://doi.org/10.1371/journal.pone.0107042

Su, M. D., Lin, M. C., Hsieh, H. I., Tsai, B. W., & Lin, C. H. (2010). Multi-layer multi-class dasymetric mapping to estimate population distribution. *Science of the Total Environment*, *408*, 4807–4816. https://doi.org/10.1016/j.scitotenv.2010.06.032

Suharto, S. (2001). Complementary Sources of Demographic and Social Statistics. *Symposium on Global Review of 2000 Round of Population and Housing Censuses: Mid-Decade Assessment and Future Prospects*.

Sun, K., Zhu, Y., Pan, P., Hou, Z., Wang, D., Li, W., & Song, J. (2019). Geospatial data ontology : the semantic foundation of geospatial data integration and sharing. *Big Earth Data*, *3*(3), 269–296. https://doi.org/10.1080/20964471.2019.1661662

Šveda, M., Hurbánek, P., Madajová, M. S., Rosina, K., Förstl, F., Záboj, P., & Výbošťok, J. (2024). When spatial interpolation matters: Seeking an appropriate data transformation from the mobile network for population estimates. *Computers, Environment and Urban Systems*, *110*. https://doi.org/10.1016/j.compenvurbsys.2024.102106

Swanwick, R. H., Read, Q. D., Guinn, S. M., Williamson, M. A., Hondula, K. L., & Elmore, A. J. (2022). Dasymetric population mapping based on US census data and 30-m gridded estimates of impervious surface. *Scientific Data*, *9*, 1–7. https://doi.org/10.1038/s41597-022-01603-z

Tao, L., Ma, K., Tian, M., Hui, Z., Zheng, S., Liu, J., Xie, Z., & Qiu, Q. (2024). Developing a Base Domain Ontology from Geoscience Report Collection to Aid in Information Retrieval towards Spatiotemporal and Topic Association. *ISPRS International Journal of Geo-Information*, *13*(1). https://doi.org/10.3390/ijgi13010014

Telega, A., Telega, I., & Bieda, A. (2021). Measuring Walkability with GIS — Methods Overview and New Approach Proposal. *Sustainability*, *13*(4). https://doi.org/https://doi.org/10.3390/su13041883

Tobler, W. R. (1979). Smooth pycnophylactic interpolation for geographical regions. *Journal of the American Statistical Association*, *74*(367), 519–530. https://doi.org/10.1080/01621459.1979.10481647

Unicode Consortium. (2024). *The Unicode Standard v16.0.0*. Retrieved 19 September 2025. https://www.unicode.org/versions/Unicode16.0.0/

United Nations Department of Economic and Social Affairs. (2005). Household Sample Surveys in Developing and Transition Countries. In *Studies in Methods: Vol. F* (Issue 96). http://unstats.un.org/unsd/hhsurveys/

United Nations Department of Economic and Social Affairs. (2008). Sources of data for social and demographic statistics. In *Designing Household Survey Samples: Practical Guidelines* (Issue 98).

United Nations Department of Economic and Social Affairs. (2017). Principles and recommendations for population and housing censuses. In *Statistical Papers* (Vol. 3, Issue March).

United Nations Department of Economic and Social Affairs. (2019). *Introduction to SDMX data modeling*. Retrieved 24 September 2025.

https://ggim.un.org/meetings/2019/Deqing/documents/S5 D Eshetie.pdf

United Nations Department of Economic and Social Affairs. (2021). *Spatial anonymization: guidance note for the Inter-Secretariat Working Group on household surveys*. https://unstats.un.org/unsd/statcom/52nd-session/documents/BG-3l-Spatial_Anonymization-E.pdf.

United Nations Department of Economic and Social Affairs. (2022). Dissemination of Official Statistics. In *The Handbook on Management and Organization of National Statistical Systems* (pp. 485–517).

United Nations Economic Commission for Europe. (2007). Register-based statistics in the Nordic countries: Review of best practices with focus on population and social statistics. In *United Nations Economic Commission for Europe*. http://www.unece.org/fileadmin/DAM/stats/publications/Register_based_statistics_in_Nordic_countries.pdf

Ural, S., Hussain, E., & Shan, J. (2011). Building population mapping with aerial imagery and GIS data. *International Journal of Applied Earth Observation and Geoinformation*, *13*(6), 841–852. https://doi.org/10.1016/j.jag.2011.06.004

Uschold, M., & Gruninger, M. (1996). Ontologies: principles, methods and applications. *The Knowledge Engineering Review*, *11*(2), 93–136. https://doi.org/doi:10.1017/S0269888900007797

W3C. (2004a). *OWL-S: Semantic Markup for Web Services*. Retrieved 24 September 2025. https://www.w3.org/submissions/OWL-S/

W3C. (2004b). *OWL Web Ontology Language*. Retrieved 20 September 2025. https://www.w3.org/TR/2004/REC-owl-features-20040210/#s2.2

W3C. (2008a). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Retrieved 19 September 2025. https://www.w3.org/TR/xml/#sec-intro

W3C. (2008b). *SPARQL Query Language for RDF*. Retrieved 20 September 2025. https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

W3C. (2012). *OWL 2 Web Ontology Language Profiles*. Retrieved 25 September

2025. https://www.w3.org/TR/owl2-profiles/#OWL_2_RL_2

W3C. (2013). *SPARQL 1.1 Update*. Retrieved 21 September 2025. https://www.w3.org/TR/sparql11-update/

W3C. (2014a). *RDF Schema 1.1*. Retrieved 20 September 2025. https://www.w3.org/TR/rdf-schema/#ch_domainrange

W3C. (2014b). *The RDF Data Cube Vocabulary*. Retrieved 23 September 2025. https://www.w3.org/TR/vocab-data-cube/

W3C. (2017). *Semantic Sensor Network Ontology*. Retrieved 24 September 2025. https://www.w3.org/TR/vocab-ssn/

W3C. (2022). *Time Ontology in OWL*. Retrieved 24 September 2025. https://www.w3.org/TR/owl-time/

W3Techs. (2025). *Usage of character encodings by ranking*. Retrieved 19 September 2025. https://w3techs.com/technologies/cross/character_encoding/ranking

Wang, S., & Wang, L. (2024). A Novel Framework for Mapping Updated Fine-resolution Populations with Remote Sensing and Mobile Phone Data. *Journal of Remote Sensing*, *4*, 1-14. https://doi.org/10.34133/remotesensing.0227

Wang, Y., Zhang, X., Lu, H., Matthews, K. A., & Greenlund, K. J. (2020). Intercensal and postcensal estimation of population size for small geographic areas in the United States. *International Journal of Population Data Science*, *5*(1), 1–8. https://doi.org/10.23889/IJPDS.V5I1.1160

Weber, E. M., Seaman, V. Y., Stewart, R. N., Bird, T. J., Tatem, A. J., McKee, J. J., Bhaduri, B. L., Moehl, J. J., & Reith, A. E. (2018). Census-independent population mapping in northern Nigeria. *Remote Sensing of Environment*, *204*, 786–798. https://doi.org/10.1016/j.rse.2017.09.024

Wieczorek, W. F., & Delmerico, A. M. (2010). Geographic information systems. *Wiley Interdisciplinary Reviews Computational Statistics*, *1*(2), 167–186. https://doi.org/https://doi.org/10.1002/wics.21

Wong, A., Fox, M., & Katsumi, M. (2024). Semantically interoperable census data:

unlocking the semantics of census datausing ontologies and linked data. *International Journal of Population Data Science*, *9*(1), 1–29. https://doi.org/https://doi.org/10.23889/ijpds.v9i1.2378

Wong, W., Liu, W., & Bennamoun, M. (2012). Ontology learning from text: A look back and into the future. *ACM Computing Surveys*, *44*(4). https://doi.org/10.1145/2333112.2333115

Worboys, M. (1992). A Model for Spatio-Tmeporal Information. *Proceedings: The 5th International Symposium on Spatial Data Handling*, 602–611.

World Pop. (2024). *WorldPop Book of Methods, Vol. I: Gridded Population Estimates*. University of Southampton. https://wpgp.github.io/bookworm/index.html

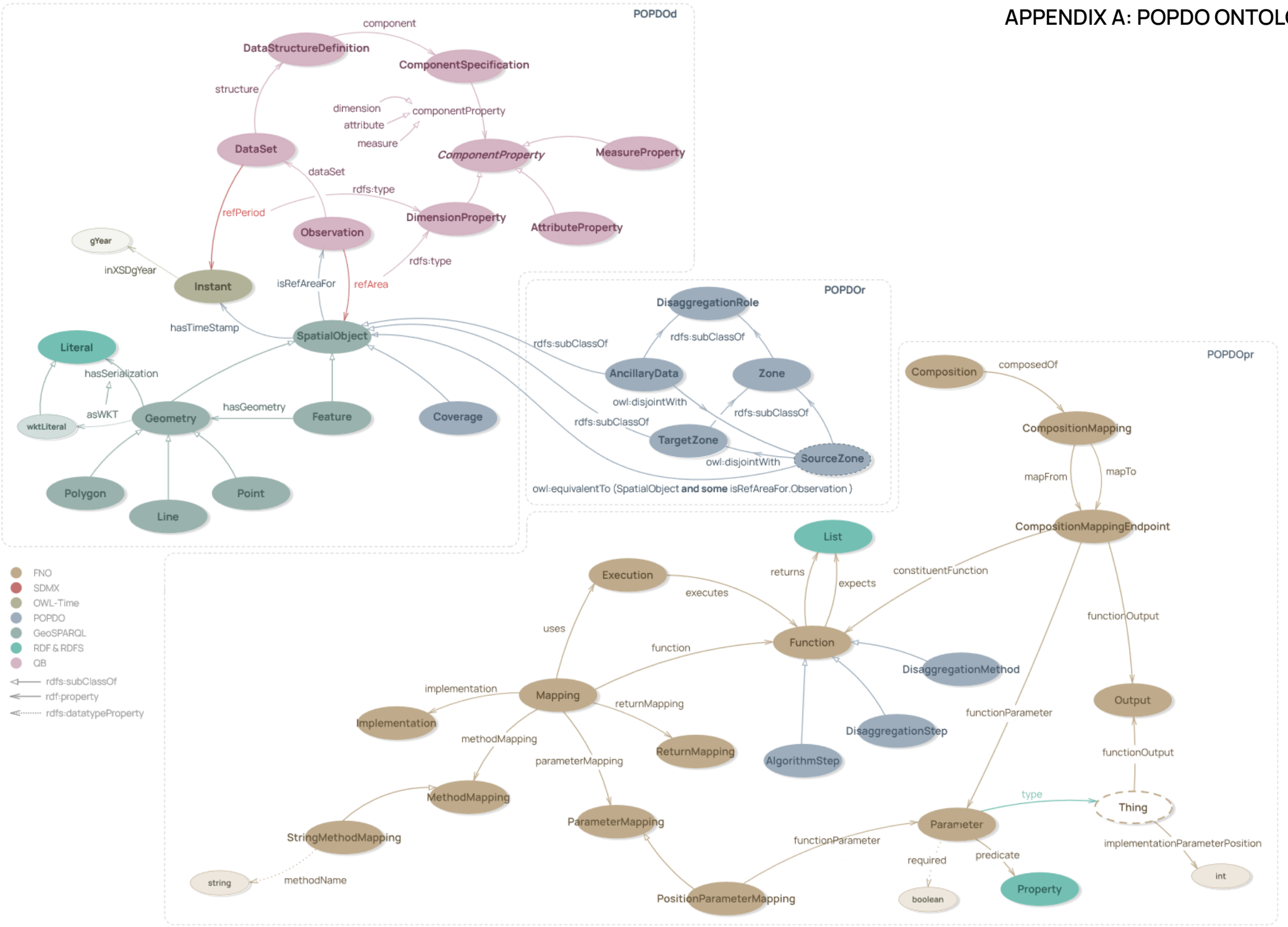
Yang, Y., & Claramunt, C. (2003). A Process-oriented Multi-representation of Gradual Changes. *Journal of Geographic Information and Decision Analysis*, *7*(1), 1–13.

Yuan, M. (1996). Temporal GIS and spatio-temporal modeling. *Third International Conference on Integrating GIS and Environmental Modeling*. http://loi.sscc.ru/gis/data_model/may.html

Yuan, M. (1997). Modelling Semantical 1 , Spatial and Temporal Information in a GIS. In M. Craglia & H. Couclelis (Eds.), *Geographic Information Research* (1st ed., p. 14). CRC Press. https://doi.org/https://doi.org/10.1201/9781003062691

APPENDICES

APPENDIX A: POPDO ONTOLOGY



APPENDIX B: DATA DESCRIPTION

Dataset	Data Source	Metadata	Attributes	License
Population by Towns/Municipalities	National Bureau of Statistics (NBS) – 2021 Census	2021 [1] Tabular data [2] NBS data portal [3] *.xslx [4]	Spatial unit name Total population	Open Government License
Natural change in population, 2022	National Bureau of Statistics (NBS)	2022 [1] Tabular data [2] NBS website [3] *.xslx [4]	Spatial unit name Population change	Open Government License
Local self-government units	State Geodetic Administration Geoportal	2025 [1] Vectors (polygon) [2] ATOM service [3] *.gml [4] ETRS89/LAEA [5]	ID Text (name) WKT geometry	Open Government License
Building Footprints	State Geodetic Administration Geoportal	2025 [1] Vectors (polygon) [2] ATOM service [3] *.gml [4] HTRS96/TM [5]	ID Use code WKT geometry	Open Government License
Area of Interest (arbitrary target zone)	User created (QGIS)	Vector (polygon) [2] HTRS96/TM [5]	WKT geometry	/

[1] reference year, [2] data type, [3] access type, [4] data format, [5] coordinate reference system

APPENDIX C: SCHEMA INDIVIDUALS

FUNCTION INDIVIDUALS

```
@base <http://www.example.com/function/> .
@prefix fun: <http://www.example.com/function/> .
@prefix popd: <http://www.example.com/ontology/popdcore#> .
@prefix bdasy: <http://www.example.com/ontology/buildingareadasy#> .
@prefix admunit: <http://www.example.com/data/admunit/> .
@prefix build: <http://www.example.com/data/building/> .
@prefix popcnt: <http://www.example.com/data/popcount/> .
@prefix popcng: <http://www.example.com/data/popchange/> .
@prefix areai: <http://www.example.com/data/areainterest/> .
@prefix fno: <https://w3id.org/function/ontology#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms/> .

#DISAGGREGATION METHOD
fun:DisaggregationMethod a popd:DisaggregationMethod ;
        fno:name "Top-level disaggregation function acting as a procedure"^^xsd:string ;
        dcterms:description "Main disaggregation workflow."^^xsd:string ;
        fno:expects (fun:SourceZoneQuery fun:ResidentialBuildingQuery
fun:ArbitraryUnitQuery) ;
        fno:returns (fun:FinalArbitraryUnit) .

fun:SourceZoneQuery a fno:Parameter ;
        rdfs:comment "Input SPARQL query for source zone filtering"@en ;
        fno:predicate fun:hasSourceZoneQuery ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:ResidentialBuildingQuery a fno:Parameter ;
        rdfs:comment "Input SPARQL query for residential buildings filtering"@en ;
        fno:predicate fun:hasResidentialBuildingQuery ;
        fno:type xsd:string;
        fno:required "true"^^xsd:boolean .

fun:ArbitraryUnitQuery a fno:Parameter ;
        rdfs:comment "Input SPARQL query for arbitrary spatial unit filtering"@en ;
        fno:predicate fun:hasArbitraryUnitQuery ;
        fno:type xsd:string;
        fno:required "true"^^xsd:boolean .
```

```
fun:FinalArbitraryUnit a fno:Output ;
        rdfs:comment "Population disaggregation result"@en ;
        fno:predicate fun:hasDisaggregatedPopulation ;
        fno:type popd:TargetZone;
        fno:required "true"^^xsd:boolean .

#DISAGGREGATION STEP (TEMPORAL ADJUSTMENT)
fun:TemporalAdjustment a popd:DisaggregationStep ;
        fno:name "Temporal population adjustment"^^xsd:string ;
        dcterms:description "Middle level function performing temporal adjustment."^^xsd:string
;
        fno:expects (fun:SourceZoneQuery2) ;
        fno:returns (fun:ResultingSourceZone1) .

fun:SourceZoneQuery2 a fno:Parameter ;
        rdfs:comment "Filter for reaching Source Zone URI"@en ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:ResultingSourceZone1 a fno:Output ;
        rdfs:comment "Adjusted population value tied to spatial"@en ;
        fno:type popd:SourceZone .

##ALGORITHM STEP (TEMPORAL ADJUSTMENT)
fun:SimplePopulationAdjustment a popd:AlgorithmStep ;
        fno:name "Temporal population adjustment"^^xsd:string ;
        dcterms:description "Bottom level function adjusting population data."^^xsd:string ;
        fno:expects (fun:SourceZoneQuery3) ;
        fno:returns (fun:OutputSourceZone1) .

fun:SourceZoneQuery3 a fno:Parameter ;
        rdfs:comment "Filter for reaching Source Zone URI"@en ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:OutputSourceZone1 a fno:Output ;
        rdfs:comment "Returning Source Zone URI"@en ;
        fno:type popd:SourceZone .

#DISAGGREGATION STEP (WEIGHT COMPUTATION)
fun:WeightComputation a popd:DisaggregationStep ;
        fno:name "Calculation of weights"^^xsd:string ;
```

```
        dcterms:description "Middle level function performing weight computation."^^xsd:string
;
        fno:expects (fun:SourceZoneInput1 fun:ResidentialBuildingQuery2) ;
        fno:returns (fun:ResultingResidentialBuilding1) .

fun:SourceZoneInput1 a fno:Parameter ;
        rdfs:comment "Source Zone as admisnistrative unit"@en ;
        fno:type popd:SourceZone ;
        fno:required "true"^^xsd:boolean .

fun:ResidentialBuildingQuery2 a fno:Parameter ;
        rdfs:comment "Filter for reaching Residential Buildings URIs"@en ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:ResultingResidentialBuilding1 a fno:Output ;
        rdfs:comment "Residential buildings with weights"@en ;
        fno:type rdfs:List .

#---
#ALGORITHM STEP (AREA RATIO, ESTIMATED DENSITY, DENSITY RATIO, TOTAL
FRACTION)
fun:AreaRatio a popd:AlgorithmStep ;
        fno:name "Building Area Ratio Function"^^xsd:string ;
        dcterms:description "Takes a query returning building features with WKT geometries.
Calculates the area for each and its ratio to the total, then attaches these values to each
building."^^xsd:string ;
        fno:expects (fun:ResidentialBuildingQuery3) ;
        fno:returns (fun:OutputResidentialBuilding1) .

fun:ResidentialBuildingQuery3 a fno:Parameter ;
        rdfs:comment "Filter for reaching Residential Buildings URIs"@en ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:OutputResidentialBuilding1 a fno:Output ;
        rdfs:comment "Returning Residential Buildings URIs"@en ;
        fno:type rdfs:List .
#----
fun:EstimatedPopulationDensity a popd:AlgorithmStep ;
        fno:name "Estimated Population Density Function"^^xsd:string ;
        dcterms:description "Function that calculates estimated population density for a specific
residential building class"^^xsd:string ;
```

```
        fno:expects (fun:SourceZoneInput2 fun:ResidentialBuildingInput1) ;
        fno:returns (fun:OutputResidentialBuilding2) .

fun:SourceZoneInput2 a fno:Parameter ;
        rdfs:comment "SourceZone URI"@en ;
        fno:type popd:SourceZone ;
        fno:required "true"^^xsd:boolean .

fun:ResidentialBuildingInput1 a fno:Parameter ;
        rdfs:comment "Residential building URI"@en ;
        fno:type rdfs:List ;
        fno:required "true"^^xsd:boolean .

fun:OutputResidentialBuilding2 a fno:Output ;
        rdfs:comment "Returning Residential Buildings URIs"@en ;
        fno:type rdfs:List .
#----
fun:DensityRatio a popd:AlgorithmStep ;
        fno:name "Population density function"^^xsd:string ;
        dcterms:description "Function that normalizes estimated population densities for every
class by dividing it with the sum of all estimated densities"^^xsd:string ;
        fno:expects (fun:ResidentialBuildingInput2) ;
        fno:returns (fun:OutputResidentialBuilding3) .

fun:ResidentialBuildingInput2 a fno:Parameter ;
        rdfs:comment "Residential building URI"@en ;
        fno:type rdfs:List ;
        fno:required "true"^^xsd:boolean .

fun:OutputResidentialBuilding3 a fno:Output ;
        rdfs:comment "Returning Residential Buildings URIs"@en ;
        fno:type rdfs:List .
#----
fun:TotalFraction a popd:AlgorithmStep ;
        fno:name "Total fraction function"^^xsd:string ;
        dcterms:description "Function that calculates ratio of areaXdensity product and sum of
all areaXdensity products"^^xsd:string ;
        fno:expects (fun:ResidentialBuildingInput3) ;
        fno:returns (fun:OutputResidentialBuilding4) .

fun:ResidentialBuildingInput3 a fno:Parameter ;
        rdfs:comment "Residential building URI"@en ;
        fno:type rdfs:List ;
```

```
        fno:required "true"^^xsd:boolean .

fun:OutputResidentialBuilding4 a fno:Output ;
        rdfs:comment "Returning Residential Buildings URIs"@en ;
        fno:type rdfs:List .

#DISAGGREGATION STEP (DISAGGREGATION)

fun:Disaggregation a popd:DisaggregationStep ;
        fno:name "Disaggregation function"^^xsd:string ;
        dcterms:description "Middle level function performing disaggregation."^^xsd:string ;
        fno:expects (fun:SourceZoneInput3 fun:ResidentialBuildingInput4) ;
        fno:returns (fun:ResultingResidentialBuilding2) .

fun:SourceZoneInput3 a fno:Parameter ;
        fno:type popd:SourceZone ;
        fno:required "true"^^xsd:boolean .

fun:ResidentialBuildingInput4 a fno:Parameter ;
        fno:type rdfs:List ;
        fno:required "true"^^xsd:boolean .

fun:ResultingResidentialBuilding2 a fno:Output ;
        rdfs:comment "Residential buildings with weights"@en ;
        fno:type rdfs:List .

#ALGORITHM STEP (SIMPLE DISAGGREGATION)
fun:SimpleDisaggregation a popd:AlgorithmStep ;
        fno:name "Simple disaggregation function"^^xsd:string ;
        dcterms:description "Function that calculated population at building level"^^xsd:string ;
        fno:expects (fun:SourceZoneInput4 fun:ResidentialBuildingInput5) ;
        fno:returns (fun:OutputResidentialBuilding5) .

fun:SourceZoneInput4 a fno:Parameter ;
        fno:type popd:SourceZone ;
        fno:required "true"^^xsd:boolean .

fun:ResidentialBuildingInput5 a fno:Parameter ;
        fno:type rdfs:List ;
        fno:required "true"^^xsd:boolean .

fun:OutputResidentialBuilding5 a fno:Output ;
        fno:type rdfs:List ;
```

```
        fno:required "true"^^xsd:boolean .

#DISAGGREGATION STEP (AGGREGATION)
fun:Aggregation a popd:DisaggregationStep ;
        fno:name "Aggregation function"^^xsd:string ;
        dcterms:description "Middle level function performing aggregation."^^xsd:string ;
        fno:expects (fun:ResidentialBuildingInput6 fun:ArbitraryUnitQuery2) ;
        fno:returns (fun:ResultingArbitraryUnit1) .

fun:ResidentialBuildingInput6 a fno:Parameter ;
        fno:type rdfs:List ;
        fno:required "true"^^xsd:boolean .

fun:ArbitraryUnitQuery2 a fno:Parameter ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:ResultingArbitraryUnit1 a fno:Output ;
        fno:type popd:TargetZone ;
        fno:required "true"^^xsd:boolean .

##ALGORITHM STEP (SIMPLE AGGREGATION)
fun:SimpleAggregation a popd:AlgorithmStep ;
        fno:name "Simple aggregation function"^^xsd:string ;
        dcterms:description "Function that calculated population at arbitrary spatial
unit"^^xsd:string ;
        fno:expects (fun:ResidentialBuildingInput7 fun:ArbitraryUnitQuery3) ;
        fno:returns (fun:ResultingArbitraryUnit2) .

fun:ResidentialBuildingInput7 a fno:Parameter ;
        fno:type rdfs:List ;
        fno:required "true"^^xsd:boolean .

fun:ArbitraryUnitQuery3 a fno:Parameter ;
        fno:type xsd:string ;
        fno:required "true"^^xsd:boolean .

fun:ResultingArbitraryUnit2 a fno:Output ;
        fno:type popd:TargetZone ;
        fno:required "true"^^xsd:boolean .
```

EXECUTION INDIVIDUAL

```
exe:DisaggregationMethodExecution a fno:Execution ;
        fno:executes fun:DisaggregationMethod ;
        fno:uses map:DisaggregationMethodMapping ;
        fun:hasSourceZoneQuery
                "PREFIX popd: <http://www.example.com/ontology/popdcore#>
                SELECT ?sourceZone
                WHERE {?sourceZone a popd:SourceZone .}"^^xsd:string ;
        fun:hasResidentialBuildingQuery
                "PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
                SELECT ?building
                WHERE {?building a bdasy:BuildingFootprint ; bdasy:hasUseCode ?useCode .
                        FILTER(?useCode IN (100, 101, 102, 103))}"^^xsd:string  ;
        fun:hasArbitraryUnitQuery
                "PREFIX popd: <http://www.example.com/ontology/popdcore#>
                SELECT ?arbitraryUnit
                WHERE {?arbitraryUnit a popd:TargetZone. }"^^xsd:string ;
        fun:hasDisaggregatedPopulation areai:AreaOfInterest .
```

MAPPING INDIVIDUALS

```
#DisaggregationMethod MAPPING
map:DisaggregationMethodMapping a fno:Mapping ;
        fno:function fun:DisaggregationMethod ;
        fno:implementation imp:DisaggregationMethodImplementation .

#TemporalAdjustment MAPPING
map:TemporalAdjustmentMapping a fno:Mapping ;
        fno:function fun:TemporalAdjustment ;
        fno:implementation imp:TemporalAdjustmentImplementation .

#WeightComputation MAPPING
map:WeightComputationMapping a fno:Mapping ;
        fno:function fun:WeightComputation ;
        fno:implementation imp:WeightComputationImplementation .

#Disaggregation MAPPING
map:DisaggregationMapping a fno:Mapping ;
        fno:function fun:Disaggregation ;
        fno:implementation imp:DisaggregationImplementation .
```

```
#Aggregation MAPPING
map:AggregationMapping a fno:Mapping ;
        fno:function fun:Aggregation ;
        fno:implementation imp:AggregationImplementation .

#TEMPORAL ADJUSTMENT
map:SimplePopulationAdjustmentMapping a fno:Mapping ;
        fno:function fun:SimplePopulationAdjustment ;
        fno:implementation imp:SimplePopulationAdjustmentImplementation ;
        fno:methodMapping [ a fnom:StringMethodMapping ;
                                fnom:method-name "doTemporalAdjustment"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:SourceZoneQuery3 ;
                                fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                                fnom:functionOutput fun:OutputSourceZone1 ] .

#AreaRatio
map:AreaRatioMapping a fno:Mapping ;
        fno:function fun:AreaRatio ;
        fno:implementation imp:AreaRatioImplementation ;
        fno:methodMapping [ a fnom:StringMethodMapping ;
                                fnom:method-name "doAreaRatio"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:ResidentialBuildingQuery3 ;
                                fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                                fnom:functionOutput fun:OutputResidentialBuilding1 ] .

#EstimatedDenstiy
map:EstimatedDensityMapping a fno:Mapping ;
        fno:function fun:EstimatedPopulationDensity ;
        fno:implementation imp:EstimatedPopulationDensityImplementation ;
        fno:methodMapping [ a fnom:StringMethodMapping ;
                                fnom:method-name "doDensityEstimation"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:SourceZoneInput2 ;
                                fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:ResidentialBuildingInput1 ;
                                fnom:implementationParameterPosition "2"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                                fnom:functionOutput fun:OutputResidentialBuilding2 ] .
```

```
#DensityRatio
map:DensityRatioMapping a fno:Mapping ;
        fno:function fun:DensityRatio ;
        fno:implementation imp:DensityRatioImplementation ;
        fno:methodMapping [ a fnom:StringMethodMapping ;
                                fnom:method-name "doDensityRatio"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:ResidentialBuildingInput2 ;
                                fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                                fnom:functionOutput fun:OutputResidentialBuilding3 ] .

#TotalFraction
map:TotalFractionMapping a fno:Mapping ;
        fno:function fun:TotalFraction ;
        fno:implementation imp:TotalFractionImplementation ;
        fno:methodMapping [ a fnom:StringMethodMapping ;
                                fnom:method-name "doTotalFraction"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:ResidentialBuildingInput3 ;
                                fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                                fnom:functionOutput fun:OutputResidentialBuilding4 ] .

#Disaggregation
map:SimpleDisaggregationMapping a fno:Mapping ;
        fno:function fun:SimpleDisaggregation ;
        fno:implementation imp:SimpleDisaggregationImplementation ;
        fno:methodMapping [ a fnom:StringMethodMapping ;
                                fnom:method-name "doDisaggregation"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:SourceZoneInput4 ;
                                fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                                fnom:functionParameter      fun:ResidentialBuildingInput5 ;
                                fnom:implementationParameterPosition "2"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                                fnom:functionOutput fun:OutputResidentialBuilding5 ] .

#Aggregation
map:SimpleAggregationMapping a fno:Mapping ;
        fno:function fun:SimpleAggregation ;
        fno:implementation imp:SimpleAggregationImplementation ;
```

```
        fno:methodMapping [ a fnom:StringMethodMapping ;
                            fnom:method-name "doArbitraryAggregation"^^xsd:string] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                            fnom:functionParameter      fun:ResidentialBuildingInput7 ;
                            fnom:implementationParameterPosition "1"^^xsd:int ] ;
        fno:parameterMapping [ a fnom:PositionParameterMapping ;
                            fnom:functionParameter      fun:ArbitraryUnitQuery3 ;
                            fnom:implementationParameterPosition "2"^^xsd:int ] ;
        fno:returnMapping [ a fnom:DefaultReturnMapping ;
                            fnom:functionOutput fun:ResultingArbitraryUnit2 ] .
```

IMPLEMENTATION INDIVIDUALS

```
#DISAGGREGATION METHOD
imp:DisaggregationMethodImplementation a fno:Implementation ;
        rdfs:label "Links to composition individual"@en ;
        imp:implements comp:DisaggregationMethodComposition ;
        rdf:comment "implementation using composition rules" .

imp:implements rdf:type owl:ObjectProperty ;
        rdfs:domain fnoi:Implementation ;
        rdfs:range fnoc:Composition .

#TEMPORAL ADJUSTMENT
imp:TemporalAdjustmentImplementation a fno:Implementation ;
        rdfs:label "Links to composition individual"@en ;
        imp:implements comp:TemporalAdjustmentComposition ;
        rdf:comment "implementation using composition rules" .

#WEIGHT COMPUTATION
imp:WeightComputationImplementation a fno:Implementation ;
        rdfs:label "Links to composition individual"@en ;
        imp:implements comp:WeightComputationComposition ;
        rdf:comment "implementation using composition rules" .

#DISAGGREGATION
imp:DisaggregationImplementation a fno:Implementation ;
        rdfs:label "Links to composition individual"@en ;
        imp:implements comp:DisaggregationComposition ;
        rdf:comment "implementation using composition rules" .

#AGGREGATION
imp:AggregationImplementation a fno:Implementation ;
```

```
        rdfs:label "Links to composition individual"@en ;
        imp:implements comp:AggregationComposition ;
        rdf:comment "implementation using composition rules" .

#Simple Population Adjustment
imp:SimplePopulationAdjustmentImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/DoTemporalAdjustment.py>;
        rdf:comment "python script for area ratio calculation" .

imp:location rdf:type owl:DatatypeProperty ;
        rdfs:domain fnoi:Implementation .

#AreaRatio
imp:AreaRatioImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/DoAreaRatio.py>;
        rdf:comment "python script for area ratio calculation" .

#EstimatedDensity
imp:EstimatedPopulationDensityImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/DoDensityEstimation.py>;
        rdf:comment "python script for calculation of estimated population density" .

#DensityRatio
imp:DensityRatioImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/DoDensityRatio.py>;
        rdf:comment "python script for density ratio calculation" .

#TotalFraction
imp:TotalFractionImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/DoTotalFraction.py>;
        rdf:comment "python script for total fraction calculation" .

#Disaggregation
imp:SimpleDisaggregationImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/doDisaggregation.py>;
        rdf:comment "python script for disaggregated population calculation" .
```

```
#Aggregacija
imp:SimpleAggregationImplementation a fno:Implementation ;
        rdfs:label "Python script"@en ;
        imp:location <file:///C:/Scripts/doArbitraryAggregation.py>;
        rdf:comment "python script for calculating population within arbitrary unit" .

COMPOSITION INDIVIDUALS

#Disaggregation Method Composition
comp:DisaggregationMethodComposition a fnoc:Composition ;
        fnoc:composedOf
        [
                # 1. Map Initial Source Query -> TemporalAdjustment
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:DisaggregationMethod ;
                        fnoc:functionParameter fun:SourceZoneQuery
                ] ;
                fnoc:mapTo [
                        fnoc:constituentFunction fun:TemporalAdjustment ;
                        fnoc:functionParameter fun:SourceZoneQuery2
                ]
        ] ,
        [
                # 2. Map Initial Building Query -> WeightComputation
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:DisaggregationMethod ;
                        fnoc:functionParameter fun:ResidentialBuildingQuery
                ] ;
                fnoc:mapTo [
                        fnoc:constituentFunction fun:WeightComputation ;
                        fnoc:functionParameter fun:ResidentialBuildingQuery2
                ]
        ] ,
        [
                # 3. Map Initial Arbitrary Unit Query -> Aggregation
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:DisaggregationMethod ;
                        fnoc:functionParameter fun:ArbitraryUnitQuery
                ] ;
                fnoc:mapTo [
                        fnoc:constituentFunction fun:Aggregation ;
                        fnoc:functionParameter fun:ArbitraryUnitQuery2
                ]
```

```
    ] ,
    [
        # 4. Connect TemporalAdjustment Output -> WeightComputation Input
        fnoc:mapFrom [
            fnoc:constituentFunction fun:TemporalAdjustment ;
            fnoc:functionOutput fun:ResultingSourceZone1
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:WeightComputation ;
            fnoc:functionParameter fun:SourceZoneInput1
        ]
    ] ,
    [
        # 5. Connect TemporalAdjustment Output -> Disaggregation Input
        fnoc:mapFrom [
            fnoc:constituentFunction fun:TemporalAdjustment ;
            fnoc:functionOutput fun:ResultingSourceZone1
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:Disaggregation ;
            fnoc:functionParameter fun:SourceZoneInput3
        ]
    ] ,
    [
        # 6. Connect WeightComputation Output -> Disaggregation Input
        fnoc:mapFrom [
            fnoc:constituentFunction fun:WeightComputation ;
            fnoc:functionOutput fun:ResultingResidentialBuilding1
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:Disaggregation ;
            fnoc:functionParameter fun:ResidentialBuildingInput4
        ]
    ] ,
    [
        # 7. Connect Disaggregation Output -> Aggregation Input
        fnoc:mapFrom [
            fnoc:constituentFunction fun:Disaggregation ;
            fnoc:functionOutput fun:ResultingResidentialBuilding2
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:Aggregation ;
            fnoc:functionParameter fun:ResidentialBuildingInput6
```

```
                ]
        ] ,
        [
                # 8. Connect Aggregation Output -> Final Workflow Output
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:Aggregation ;
                        fnoc:functionOutput fun:ResultingArbitraryUnit1
                ] ;
                fnoc:mapTo [
                        fnoc:constituentFunction fun:DisaggregationMethod ;
                        fnoc:functionOutput fun:FinalArbitraryUnit
                ]
        ].

#Temporal Adjustment Composition
comp:TemporalAdjustmentComposition a fnoc:Composition ;
        fnoc:composedOf [
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:TemporalAdjustment ;
                        fnoc:functionParameter fun:SourceZoneQuery2
                ] ;
                fnoc:mapTo [
                        fnoc:constituentFunction fun:SimplePopulationAdjustment ;
                        fnoc:functionParameter fun:SourceZoneQuery3
                ]
        ] ,
        [
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:SimplePopulationAdjustment ;
                        fnoc:functionOutput fun:OutputSourceZone1
                ] ;
                fnoc:mapTo [
                        fnoc:constituentFunction fun:TemporalAdjustment ;
                        fnoc:functionOutput fun:ResultingSourceZone1
                ]
        ] .

#WeightComputation Composition
comp:WeightComputationComposition a fnoc:Composition ;
        fnoc:composedOf [
                fnoc:mapFrom [
                        fnoc:constituentFunction fun:WeightComputation ;
                        fnoc:functionParameter fun:SourceZoneInput1
```

```
        ] ;
        fnoc:mapTo [
                fnoc:constituentFunction fun:EstimatedPopulationDensity ;
                fnoc:functionParameter fun:SourceZoneInput2
        ]
] ,
[
        fnoc:mapFrom [
                fnoc:constituentFunction fun:WeightComputation ;
                fnoc:functionParameter fun:ResidentialBuildingQuery2
        ] ;
        fnoc:mapTo [
                fnoc:constituentFunction fun:AreaRatio ;
                fnoc:functionParameter fun:ResidentialBuildingQuery3
        ]
] ,
[
        fnoc:mapFrom [
                fnoc:constituentFunction fun:AreaRatio ;
                fnoc:functionOutput fun:OutputResidentialBuilding1
        ] ;
        fnoc:mapTo [
                fnoc:constituentFunction fun:EstimatedPopulationDensity ;
                fnoc:functionParameter fun:ResidentialBuildingInput1
        ]
] ,
[
        fnoc:mapFrom [
                fnoc:constituentFunction fun:EstimatedPopulationDensity ;
                fnoc:functionOutput fun:OutputResidentialBuilding2
        ] ;
        fnoc:mapTo [
                fnoc:constituentFunction fun:DensityRatio ;
                fnoc:functionParameter fun:ResidentialBuildingInput2
        ]
],
[
        fnoc:mapFrom [
                fnoc:constituentFunction fun:DensityRatio ;
                fnoc:functionOutput fun:OutputResidentialBuilding3
        ] ;
        fnoc:mapTo [
                fnoc:constituentFunction fun:TotalFraction ;
```

```
                fnoc:functionParameter fun:ResidentialBuildingInput3
        ]
    ],
    [
        fnoc:mapFrom [
            fnoc:constituentFunction fun:TotalFraction ;
            fnoc:functionOutput fun:OutputResidentialBuilding4
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:WeightComputation ;
            fnoc:functionOutput fun:ResultingResidentialBuilding1
        ]
    ].

#Disaggregation Composition
comp:DisaggregationComposition a fnoc:Composition ;
    fnoc:composedOf [
        fnoc:mapFrom [
            fnoc:constituentFunction fun:Disaggregation ;
            fnoc:functionParameter fun:SourceZoneInput3
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:SimpleDisaggregation ;
            fnoc:functionParameter fun:SourceZoneInput4
        ]
    ] ,
    [
        fnoc:mapFrom [
            fnoc:constituentFunction fun:Disaggregation ;
            fnoc:functionParameter fun:ResidentialBuildingInput4
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:SimpleDisaggregation ;
            fnoc:functionParameter fun:ResidentialBuildingInput5
        ]
    ] ,
    [
        fnoc:mapFrom [
            fnoc:constituentFunction fun:SimpleDisaggregation ;
            fnoc:functionOutput fun:OutputResidentialBuilding5
        ] ;
        fnoc:mapTo [
            fnoc:constituentFunction fun:Disaggregation ;
```

```
                fnoc:functionOutput fun:ResultingResidentialBuilding2
            ]
        ] .

#Aggregation Composition
comp:AggregationComposition a fnoc:Composition ;
        fnoc:composedOf [
            fnoc:mapFrom [
                fnoc:constituentFunction fun:Aggregation ;
                fnoc:functionParameter fun:ResidentialBuildingInput6
            ] ;
            fnoc:mapTo [
                fnoc:constituentFunction fun:SimpleAggregation ;
                fnoc:functionParameter fun:ResidentialBuildingInput7
            ]
        ] ,
        [
            fnoc:mapFrom [
                fnoc:constituentFunction fun:Aggregation ;
                fnoc:functionParameter fun:ArbitraryUnitQuery2
            ] ;
            fnoc:mapTo [
                fnoc:constituentFunction fun:SimpleAggregation ;
                fnoc:functionParameter fun:ArbitraryUnitQuery3
            ]
        ] ,
        [
            fnoc:mapFrom [
                fnoc:constituentFunction fun:SimpleAggregation ;
                fnoc:functionOutput fun:ResultingArbitraryUnit2
            ] ;
            fnoc:mapTo [
                fnoc:constituentFunction fun:Aggregation ;
                fnoc:functionOutput fun:ResultingArbitraryUnit1
            ]
        ] .
```

APPENDIX D: PYTHON SCRIPTS

ORCHESTRATION SCRIPT

```
import sys
import importlib.util
from pathlib import Path
from urllib.parse import urlparse, unquote
from SPARQLWrapper import SPARQLWrapper, JSON
from collections import deque

# ============================================================================
# CONFIGURATION
# ============================================================================
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
TOP_LEVEL_EXECUTION_URI                                                    =
"http://www.example.com/execution/DisaggregationMethodExecution"

# ============================================================================
# DYNAMIC ORCHESTRATOR
# ============================================================================

class SimpleOrchestrator:
    """
    A recursive orchestrator that executes a workflow described by BDASY.
    """

    def __init__(self, endpoint):
        self.sparql = SPARQLWrapper(endpoint)
        self.sparql.setReturnFormat(JSON)
        self.execution_cache = {}
        self.parameter_order_cache = {}  # Cache for ordered parameters
        print(f"Orchestrator initialized for endpoint: {endpoint}")

    def _run_query(self, query):
        """Helper function to execute a SPARQL query and return bindings."""
        self.sparql.setQuery(query)
        try:
            return self.sparql.query().convert()["results"]["bindings"]
        except Exception as e:
            print(f"FATAL SPARQL Query Failed: {e}\nQuery was:\n{query}")
            raise
```

```
    def execute_function(self, function_uri, arguments):
        """Main recursive execution engine for any function."""
        if function_uri in self.execution_cache and not arguments:
            print(f"  -> Returning cached result for <{Path(function_uri).name}>")
            return self.execution_cache[function_uri]

        print(f"\nExecuting function: <{Path(function_uri).name}>")
        impl_details = self._get_implementation_details(function_uri)

        result = None
        if impl_details ['type'] == 'script':
            result = self._execute_script(impl_details, arguments)
        elif impl_details['type'] == 'composition':
            result = self._execute_composition(impl_details['uri'], arguments)

        # Only cache results of functions that don't take arguments, to be safe
        if not arguments:
            self.execution_cache[function_uri] = result
        print(f"  -> Finished executing <{Path(function_uri).name}>")
        return result

    def _get_implementation_details(self, function_uri):
        """Finds if a function is implemented by a script or a composition."""
        query = f"""
        PREFIX fno: <https://w3id.org/function/ontology#>
        PREFIX imp: <http://www.example.com/implementation/>
        PREFIX fnom: <https://w3id.org/function/vocabulary/mapping#>
        SELECT ?impl_type ?composition_uri ?script_path ?method_name WHERE {{
            ?mapping fno:function <{function_uri}> ; fno:implementation ?impl .
            OPTIONAL {{ ?impl imp:implements ?composition_uri . BIND("composition" AS
?impl_type) }}
            OPTIONAL {{
                ?impl imp:location ?script_path .
                ?mapping fno:methodMapping [ fnom:method-name ?method_name ] .
                BIND("script" AS ?impl_type)
            }}
        }} LIMIT 1
        """
        results = self._run_query(query)
        if not results: raise LookupError(f"No implementation/mapping found for <{function_uri}>")
        details = results[0]
        impl_type = details.get('impl_type', {}).get('value')
        if impl_type == 'composition':
```

```
            return {"type": "composition", "uri": details['composition_uri']['value']}
        elif impl_type == 'script':
            script_uri = details['script_path']['value']
            parsed_uri = urlparse(unquote(script_uri))
            local_path_str = parsed_uri.path

            # Fix for Windows paths: remove the first '/' from /C:/...
            if sys.platform == "win32" and local_path_str.startswith('/'):
                local_path_str = local_path_str[1:]

            local_path = Path(local_path_str)

            return {"type": "script", "path": local_path, "method": details['method_name']['value']}
        else:
            raise TypeError(f"Unknown implementation for <{function_uri}>")

    def _execute_script(self, script_details, arguments):
        """Loads and runs a Python script from a local file."""
        path, method_name = script_details['path'], script_details['method']
        print(f"   -> Running script: {path.name} -> {method_name}() with {len(arguments)}
argument(s)")
        if not path.exists(): raise FileNotFoundError(f"Script file not found: {path}")
        spec = importlib.util.spec_from_file_location(path.stem, path)
        module = importlib.util.module_from_spec(spec)
        sys.modules[spec.name] = module
        spec.loader.exec_module(module)
        method_to_call = getattr(module, method_name)
        return method_to_call(*arguments)

    def _execute_composition(self, composition_uri, composition_arguments):
        """
        Parses a composition and executes its internal functions using a robust
        topological sort for dependency resolution.
        """
        # Get the main "wrapper" function that this composition implements.
        # This is crucial for filtering.
        wrapper_func_uri = self._get_function_for_composition(composition_uri)

        # --- GRAPH BUILDING (FIXED LOGIC) ---
        adj = {}
        in_degree = {}

        # **FIX 1: Get ALL constituent functions and EXCLUDE the wrapper function.**
```

```
    # The graph should only contain the internal steps.
    all_funcs_query = f"""
    PREFIX fnoc: <https://w3id.org/function/vocabulary/composition#>
    SELECT DISTINCT ?func WHERE {{
      <{composition_uri}> fnoc:composedOf [ ?p [ fnoc:constituentFunction ?func ] ] .
      FILTER(?func != <{wrapper_func_uri}>)
    }}
    """
    all_internal_funcs = {f['func']['value'] for f in self._run_query(all_funcs_query)}
    for func in all_internal_funcs:
      adj.setdefault(func, [])
      in_degree.setdefault(func, 0)

    internal_links_query = f"""
    PREFIX fnoc: <https://w3id.org/function/vocabulary/composition#>
    SELECT DISTINCT ?source ?target WHERE {{
      <{composition_uri}> fnoc:composedOf [
        fnoc:mapFrom [ fnoc:constituentFunction ?source ] ;
        fnoc:mapTo   [ fnoc:constituentFunction ?target ]
      ] .
      FILTER(?source != <{wrapper_func_uri}> && ?target != <{wrapper_func_uri}>)
    }}
    """
    links = self._run_query(internal_links_query)
    for link in links:
      source, target = link['source']['value'], link['target']['value']
      if source in adj and target in adj:  # Ensure both are internal funcs
        adj[source].append(target)
        in_degree[target] += 1

    # --- TOPOLOGICAL SORT EXECUTION ---
    execution_queue = deque([f for f in all_internal_funcs if in_degree.get(f, 0) == 0])
    internal_output_context = {}
    executed_count = 0

    while execution_queue:
      func_uri = execution_queue.popleft()

      print(f"  -- Preparing to run internal function: <{Path(func_uri).name}>")

      # Use the robust helper to get the parameters in the correct order
      ordered_params = self._get_ordered_parameters(func_uri)
      resolved_args = []
```

```
            for param_uri in ordered_params:
                source = self._get_source_for_parameter(composition_uri, func_uri, param_uri)
                if source['type'] == 'main_argument':
                    # Get ordered params for the WRAPPER to find the index
                    wrapper_params = self._get_ordered_parameters(wrapper_func_uri)
                    arg_pos = wrapper_params.index(source['param_uri'])
                    resolved_args.append(composition_arguments[arg_pos])
                elif source['type'] == 'internal_output':
                    resolved_args.append(internal_output_context[source['source_function']])
            if "AreaRatio" in func_uri:
                print("\n\n" + "=" * 50)
                print("DEBUG: ARGUMENTS BEING SENT TO 'AreaRatio'")
                print(resolved_args)
                print("=" * 50 + "\n\n")
            output = self.execute_function(func_uri, resolved_args)
            internal_output_context[func_uri] = output
            executed_count += 1

            for next_func in adj.get(func_uri, []):
                in_degree[next_func] -= 1
                if in_degree[next_func] == 0:
                    execution_queue.append(next_func)

        if executed_count < len(all_internal_funcs):
            raise RuntimeError(
                "Workflow execution failed: A cycle was detected or the graph is disconnected."
            )

        return self._get_composition_final_output(composition_uri, internal_output_context,
wrapper_func_uri)

    def _get_source_for_parameter(self, comp_uri, target_func, target_param):
        """Finds where an input parameter for an internal function should come from."""
        query = f"""
            PREFIX fnoc: <https://w3id.org/function/vocabulary/composition#>
            SELECT ?source_func ?source_param ?source_output
            WHERE {{
                <{comp_uri}> fnoc:composedOf [
                    fnoc:mapTo   [ fnoc:constituentFunction <{target_func}> ; fnoc:functionParameter
<{target_param}> ] ;

                    # We assign the blank node to the variable ?mapFromNode
```

```
                fnoc:mapFrom ?mapFromNode
            ] .

            ?mapFromNode fnoc:constituentFunction ?source_func .
            OPTIONAL {{ ?mapFromNode fnoc:functionParameter ?source_param . }}
            OPTIONAL {{ ?mapFromNode fnoc:functionOutput ?source_output . }}

        }} LIMIT 1
        """
    results = self._run_query(query)
    if not results: raise LookupError(f"Could not find source for param <{target_param}>")

    source = results[0]
    source_func = source['source_func']['value']
    wrapper_func_uri = self._get_function_for_composition(comp_uri)

    if source_func == wrapper_func_uri:
        return {'type': 'main_argument', 'param_uri': source['source_param']['value']}
    else:
        return {'type': 'internal_output', 'source_function': source_func}

def _get_composition_final_output(self, composition_uri, internal_output_context,
wrapper_func_uri):
    """Gets the final result by finding which internal function provides the wrapper's output."""
    query = f"""
    PREFIX fnoc: <https://w3id.org/function/vocabulary/composition#>
    SELECT ?source_func WHERE {{
        <{composition_uri}> fnoc:composedOf [
            fnoc:mapTo [ fnoc:constituentFunction <{wrapper_func_uri}> ] ;
            fnoc:mapFrom [ fnoc:constituentFunction ?source_func ]
        ] .
    }} LIMIT 1
    """
    results = self._run_query(query)
    if not results: raise LookupError(f"Composition <{composition_uri}> has no final output
mapping.")

    final_source_func = results[0]['source_func']['value']
    if final_source_func not in internal_output_context:
        raise KeyError(f"The final output function <{Path(final_source_func).name}> was never
executed.")
    return internal_output_context[final_source_func]
```

```
    def _get_ordered_parameters(self, function_uri):
        """Reliably gets the ordered list of parameters for ANY function."""
        if function_uri in self.parameter_order_cache:
            return self.parameter_order_cache[function_uri]

        # The canonical way to define parameter order is via the rdf:List in fno:expects.
        query = f"""
        PREFIX fno: <https://w3id.org/function/ontology#>
        PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
        SELECT ?param WHERE {{
          <{function_uri}> fno:expects/rdf:rest*/rdf:first ?param .
        }}
        """
        # Note: This query returns parameters in the order they appear in the list.
        params = [p['param']['value'] for p in self._run_query(query)]
        self.parameter_order_cache[function_uri] = params
        return params

    def _get_function_for_composition(self, composition_uri):
        """Helper to find which main function a composition implements."""
        query = f"""
        PREFIX fno: <https://w3id.org/function/ontology#>
        PREFIX imp: <http://www.example.com/implementation/>
        SELECT ?func WHERE {{
            ?impl imp:implements <{composition_uri}> .
            ?map fno:implementation ?impl ; fno:function ?func .
        }} LIMIT 1"""
        results = self._run_query(query)
        if not results: raise LookupError(f"Could not find function for composition
<{composition_uri}>")
        return results[0]['func']['value']

    def start_from_execution(self, execution_uri):
        """The main entry point. Kicks off the workflow from an fno:Execution instance."""
        print(f"--- Starting workflow from Execution <{execution_uri}> ---")
        func_query = f"PREFIX fno: <https://w3id.org/function/ontology#> SELECT ?func
WHERE {{ <{execution_uri}> fno:executes ?func . }}"
        func_results = self._run_query(func_query)
        if not func_results: raise LookupError(f"Execution <{execution_uri}> does not specify
`fno:executes`.")
        top_level_function_uri = func_results[0]['func']['value']

        ordered_params = self._get_ordered_parameters(top_level_function_uri)
```

```
        initial_arguments = []
        for param_uri in ordered_params:
            predicate_query = f"PREFIX fno: <https://w3id.org/function/ontology#> SELECT ?pred
WHERE {{ <{param_uri}> fno:predicate ?pred . }}"
            pred_results = self._run_query(predicate_query)
            if not pred_results: raise LookupError(f"Parameter <{param_uri}> does not define
`fno:predicate`.")
            predicate = pred_results[0]['pred']['value']

            value_query = f"SELECT ?val WHERE {{ <{execution_uri}> <{predicate}> ?val . }}"
            val_results = self._run_query(value_query)
            if not val_results: raise ValueError(
                f"Execution <{execution_uri}> is missing value for predicate <{predicate}>.")
            value = val_results[0]['val']['value']
            initial_arguments.append(value)

        print(f"  -> Found initial arguments: {initial_arguments}")

        final_result = self.execute_function(top_level_function_uri, initial_arguments)

        print("\n--- WORKFLOW COMPLETE ---")
        print(f"Final Result: {final_result}")
        return final_result


# ======================================================================
# MAIN EXECUTION BLOCK
# ======================================================================

if __name__ == "__main__":
    try:
        orchestrator = SimpleOrchestrator(SPARQL_QUERY_ENDPOINT)
        orchestrator.start_from_execution(TOP_LEVEL_EXECUTION_URI)
    except Exception as e:
        print(f"\n!!!!!!!!!! ORCHESTRATION FAILED !!!!!!!!!!")
        print(f"ERROR: {e}")
        import traceback

        traceback.print_exc()
```

ALGORITHM SCRIPTS

TEMPORAL ADJUSTMENT

```
from SPARQLWrapper import SPARQLWrapper, JSON, POST

SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9/statements"

def doTemporalAdjustment(source_zone_query):
    """
    Executes a query to find a source zone, calculates a dummy adjusted population, inserts it, and returns the SINGLE URI of the processed source zone.
    """
    sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
    sparql.setReturnFormat(JSON)
    sparql.setMethod(POST)
    sparql.setQuery(source_zone_query)

    results = sparql.query().convert()["results"]["bindings"]
    if not results:
        raise ValueError("Source zone query returned no results.")

    source_zone_uri = results[0]['sourceZone']['value']

    #Placeholder for a real temporal adjustment calculation
    adjusted_population = 12312.0
    print(f"Adjusted population for {source_zone_uri} = {adjusted_population}")

    # Insert the new data
    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)
    insert_query = f"""
    PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
    PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
    INSERT DATA {{
        <{source_zone_uri}> bdasy:hasAdjustedPopulation "{adjusted_population}"^^xsd:double .
    }}
    """
    update_sparql.setQuery(insert_query)
    update_sparql.query()
```

```
    # Return a single string, not a list
    return source_zone_uri
```

AREA RATIO

```
import sys
import uuid
from SPARQLWrapper import SPARQLWrapper, JSON, POST
from shapely.wkt import loads

# Define SPARQL endpoints as constants
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT                                                    =
"http://localhost:7200/repositories/DisaggregationProba9/statements"
DATA_INSERT_CHUNK_SIZE = 500  # For inserting area/ratio data
LIST_INSERT_CHUNK_SIZE = 500  # For creating the rdfs:List itself


def create_rdfs_list_in_chunks(items, sparql_update_endpoint):
    """
    Creates an rdfs:List in the triplestore using chunked INSERTs to avoid POST size limits, and
returns its head URI.
    """
    if not items:
        return "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"

    # --- Step 1: Pre-generate all unique URIs for the list nodes ---
    base_uri = "http://www.example.com/data/list/"
    list_node_uris = [f"<{base_uri}{uuid.uuid4()}>" for _ in range(len(items))]
    head_uri = list_node_uris[0]

    # --- Step 2: Build all triples in memory first ---
    all_triples = []
    for i, node_uri in enumerate(list_node_uris):
        # Add the rdf:first triple (the actual item)
        item_uri = f"<{items[i]}>"
        all_triples.append(f"{node_uri}          <http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
{item_uri} .")

        # Add the rdf:rest triple (link to the next node or rdf:nil)
        next_node_uri = list_node_uris[i + 1] if i + 1 < len(
```

```
            list_node_uris) else "<http://www.w3.org/1999/02/22-rdf-syntax-ns#nil>"
        all_triples.append(f"{node_uri}          <http://www.w3.org/1999/02/22-rdf-syntax-ns#rest>
{next_node_uri} .")

    # --- Step 3: Insert the triples in chunks ---
    update_sparql = SPARQLWrapper(sparql_update_endpoint)
    update_sparql.setMethod(POST)

    print(f"  -> Inserting {len(all_triples)} list triples in chunks...")
    for i in range(0, len(all_triples), LIST_INSERT_CHUNK_SIZE * 2):  # *2 because each item
creates 2 triples
        chunk = all_triples[i:i + LIST_INSERT_CHUNK_SIZE * 2]
        insert_query = f"INSERT DATA {{ {' '.join(chunk)} }}"
        update_sparql.setQuery(insert_query)
        update_sparql.query()

    return head_uri.strip('<>')  # Return the URI as a plain string

def doAreaRatio(building_query):
    sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
    sparql.setReturnFormat(JSON)
    sparql.setMethod(POST)

    print("Step 1: Fetching the list of building URIs...")
    sparql.setQuery(building_query)
    results_buildings = sparql.query().convert()["results"]["bindings"]
    building_uris = [row['building']['value'] for row in results_buildings]

    if not building_uris:
        print("  -> No buildings found. Nothing to process.")
        return "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"
    print(f"  -> Found {len(building_uris)} buildings to process.")

    print("Step 2: Fetching all geometries in a single batch query...")
    values_clause = " ".join([f"<{uri}>" for uri in building_uris])
    single_geom_query = f"PREFIX geo: <http://www.opengis.net/ont/geosparql#> SELECT
?building ?wkt WHERE {{ VALUES ?building {{ {values_clause} }} ?building
geo:hasGeometry/geo:asWKT ?wkt . }}"
    sparql.setQuery(single_geom_query)
    geometry_results = sparql.query().convert()["results"]["bindings"]
    wkt_lookup = {row['building']['value']: row['wkt']['value'] for row in geometry_results}
    print(f"  -> Successfully retrieved {len(wkt_lookup)} geometries.")
```

```
    print("Step 3: Calculating areas in memory...")
    building_areas = []
    total_area = 0.0
    for uri in building_uris:
        if uri in wkt_lookup:
            wkt = wkt_lookup[uri]
            try:
                area = loads(wkt).area
                building_areas.append({'uri': uri, 'area': area})
                total_area += area
            except Exception as e:
                print(f"  -> WARNING: Could not calculate area for <{uri}>. Error: {e}")

    if total_area == 0:
        print("  -> Total area is zero. Cannot calculate ratios. Aborting.")
        return "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"
    print(f"  -> Total calculated area: {total_area}")

    print(f"Step 4: Preparing to insert data in chunks of {DATA_INSERT_CHUNK_SIZE}
buildings...")
    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)
    for i in range(0, len(building_areas), DATA_INSERT_CHUNK_SIZE):
        chunk = building_areas[i:i + DATA_INSERT_CHUNK_SIZE]
        print(f"  -> Processing data chunk {i // DATA_INSERT_CHUNK_SIZE + 1}...")
        insert_triples = []
        for building in chunk:
            ratio = building['area'] / total_area
            uri = building['uri']
            area = building['area']
            area_triple = f'<{uri}> <http://www.example.com/ontology/buildingareadasy#hasArea>
"{area}"^^<http://www.w3.org/2001/XMLSchema#double> .'
            ratio_triple = f'<{uri}>
<http://www.example.com/ontology/buildingareadasy#hasAreaRatio>
"{ratio}"^^<http://www.w3.org/2001/XMLSchema#double> .'
            insert_triples.append(area_triple)
            insert_triples.append(ratio_triple)
        insert_query_body = "\n".join(insert_triples)
        batch_insert_query = f"INSERT DATA {{ {insert_query_body} }}"
        update_sparql.setQuery(batch_insert_query)
        update_sparql.query()
    print("  -> All data chunks inserted successfully.")
```

```
    print("Step 5: Creating new rdfs:List for the processed buildings...")
    new_list_uri = create_rdfs_list_in_chunks(building_uris, SPARQL_UPDATE_ENDPOINT)
    print(f"  -> New list created: <{new_list_uri}>")

    return new_list_uri
```

DENSITY ESTIMATION

```
import sys
from SPARQLWrapper import SPARQLWrapper, JSON, POST

# Define SPARQL endpoints as constants
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT =
"http://localhost:7200/repositories/DisaggregationProba9/statements"
INSERT_CHUNK_SIZE = 500  # Number of buildings to insert per batch

def unpack_rdfs_list(list_head_uri, sparql_endpoint):
    """
    Takes the URI of the head of an rdfs:List and queries the graph
    to return a standard Python list of its members' URIs.
    """
    sparql = SPARQLWrapper(sparql_endpoint)
    sparql.setReturnFormat(JSON)
    query = f"""
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    SELECT ?member WHERE {{
      <{list_head_uri}> rdf:rest* / rdf:first ?member .
    }}
    """
    sparql.setQuery(query)
    results = sparql.query().convert()
    if not results["results"]["bindings"]:
        return []
    return [item["member"]["value"] for item in results["results"]["bindings"]]

def doDensityEstimation(source_zone_uri, building_list_uri):
    """
    Calculates the overall population density for a source zone and attaches this value to every
building in the provided list.
    """
```

```
#Ensure source_zone_uri is a string, not a list ---
if isinstance(source_zone_uri, list) and source_zone_uri:
    source_zone_uri = source_zone_uri[0]  # Take the first element
# ----------------------------------------------------------------
# Define namespaces
bdasy = "http://www.example.com/ontology/buildingareadasy#"

# --- Step 0: Unpack the rdfs:List to get a Python list of building URIs ---
print("Step 0: Unpacking the rdfs:List...")
building_footprints = unpack_rdfs_list(building_list_uri, SPARQL_QUERY_ENDPOINT)
if not building_footprints:
    print("  -> Input list is empty. Nothing to process.")
    return building_list_uri  # Return the original URI as required
print(f"  -> Found {len(building_footprints)} buildings in the list.")

sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
sparql.setReturnFormat(JSON)
sparql.setMethod(POST)  # Use POST for potentially large queries

# --- Step 1: Get the adjusted population from the SourceZone ---
print(f"Step 1: Fetching adjusted population from <{source_zone_uri}>...")
query_pop = f"""
    PREFIX bdasy: <{bdasy}>
    SELECT ?adjusted WHERE {{
        <{source_zone_uri}> bdasy:hasAdjustedPopulation ?adjusted .
    }}
    """
sparql.setQuery(query_pop)
results_pop = sparql.query().convert()
if not results_pop["results"]["bindings"]:
    raise ValueError(f"No adjusted population found for sourceZone <{source_zone_uri}>")
adjusted_population = float(results_pop["results"]["bindings"][0]["adjusted"]["value"])
print(f"  -> Adjusted population: {adjusted_population}")

# --- Step 2: Get the area for all buildings in a single query ---
print("Step 2: Getting total building area in a single batch query...")

values_clause = " ".join([f"<{uri}>" for uri in building_footprints])
query_area = f"""
    PREFIX bdasy: <{bdasy}>
    SELECT (SUM(?area) as ?totalArea)
    WHERE {{
        VALUES ?building {{ {values_clause} }}
```

```
            ?building bdasy:hasArea ?area .
        }}
        """
    sparql.setQuery(query_area)
    results_area = sparql.query().convert()

    # Initialize total_area to 0
    total_area = 0.0
    if results_area["results"]["bindings"]:
        total_area_str = results_area["results"]["bindings"][0].get("totalArea", {}).get("value")
        if total_area_str:
            total_area = float(total_area_str)

    print(f"  -> Total area: {total_area}")

    # --- Step 3: Calculate population density ---
    print("Step 3: Calculating population density...")
    density = 0.0
    if total_area > 0:
        density = adjusted_population / total_area
    print(f"  -> Calculated density: {density}")

    # --- Step 4: Attach population density to every building in chunks ---
    print(f"Step 4: Preparing to insert density data in chunks of {INSERT_CHUNK_SIZE}
buildings...")

    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)

    for i in range(0, len(building_footprints), INSERT_CHUNK_SIZE):
        chunk = building_footprints[i:i + INSERT_CHUNK_SIZE]
        print(f"  -> Processing chunk {i // INSERT_CHUNK_SIZE + 1}...")

        insert_triples = []
        for building_uri in chunk:
            density_triple = f'<{building_uri}> bdasy:hasPopulationDensity
"{density}"^^<http://www.w3.org/2001/XMLSchema#double> .'
            insert_triples.append(density_triple)

        insert_query_body = "\n".join(insert_triples)
        batch_insert_query = f"PREFIX bdasy: <{bdasy}>\nINSERT DATA {{ {insert_query_body}
}}"
```

```
        update_sparql.setQuery(batch_insert_query)
        update_sparql.query()

    print("  -> All chunks inserted successfully.")
    return building_list_uri
```

DENSITY RATIO

```
import sys
from SPARQLWrapper import SPARQLWrapper, JSON, POST

# Define SPARQL endpoints as constants
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT =
"http://localhost:7200/repositories/DisaggregationProba9/statements"
INSERT_CHUNK_SIZE = 500  # Number of buildings to insert per batch

def unpack_rdfs_list(list_head_uri, sparql_endpoint):
    """
    Takes the URI of the head of an rdfs:List and queries the graph
    to return a standard Python list of its members' URIs.
    """
    if list_head_uri == "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil":
        return []

    sparql = SPARQLWrapper(sparql_endpoint)
    sparql.setReturnFormat(JSON)
    query = f"""
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    SELECT ?member WHERE {{ <{list_head_uri}> rdf:rest* / rdf:first ?member . }}
    """
    sparql.setQuery(query)
    results = sparql.query().convert()
    if not results["results"]["bindings"]:
        return []
    return [item["member"]["value"] for item in results["results"]["bindings"]]

def doDensityRatio(building_list_uri):
    """
    Calculates the normalized density ratio for each building in a list.
    """
```

```
# --- Step 0: Unpack the list of buildings ---
print("Step 0: Unpacking the rdfs:List...")
building_uris = unpack_rdfs_list(building_list_uri, SPARQL_QUERY_ENDPOINT)
if not building_uris:
    print("  -> Input list is empty or invalid. Nothing to process.")
    return building_list_uri

print(f"  -> Found {len(building_uris)} buildings in the list.")

sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
sparql.setReturnFormat(JSON)
sparql.setMethod(POST)

# --- Step 1: Fetch all population densities in a single query ---
print("Step 1: Fetching all population densities in a single batch query...")
values_clause = " ".join([f"<{uri}>" for uri in building_uris])
density_query = f"""
PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
SELECT ?building ?density
WHERE {{
    VALUES ?building {{ {values_clause} }}
    ?building bdasy:hasPopulationDensity ?density .
}}
"""
sparql.setQuery(density_query)
results = sparql.query().convert()["results"]["bindings"]

# --- Step 2: Calculate the sum of densities in Python ---
print("Step 2: Calculating sum of densities in memory...")
building_densities = []
total_density = 0.0
for row in results:
    density = float(row['density']['value'])
    building_densities.append({'uri': row['building']['value'], 'density': density})
    total_density += density

if total_density == 0:
    print("  -> Total density is zero. Cannot calculate ratios. Aborting.")
    return building_list_uri

print(f"  -> Total density sum: {total_density}")

# --- Step 3: Insert the new density ratios in chunks ---
```

```
    print(f"Step 3: Preparing to insert density ratios in chunks of {INSERT_CHUNK_SIZE}
buildings...")
    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)

    for i in range(0, len(building_densities), INSERT_CHUNK_SIZE):
        chunk = building_densities[i:i + INSERT_CHUNK_SIZE]
        print(f"  -> Processing chunk {i // INSERT_CHUNK_SIZE + 1}...")

        insert_triples = []
        for building in chunk:
            # Calculate the normalized ratio
            ratio = building['density'] / total_density

            ratio_triple = f'<{building["uri"]}>
<http://www.example.com/ontology/buildingareadasy#hasDensityRatio>
"{ratio}"^^<http://www.w3.org/2001/XMLSchema#double> .'
            insert_triples.append(ratio_triple)

        insert_query_body = "\n".join(insert_triples)
        batch_insert_query = f"INSERT DATA {{ {insert_query_body} }}"

        update_sparql.setQuery(batch_insert_query)
        update_sparql.query()

    print("  -> All chunks inserted successfully.")

    # Return the original list URI for the next step in the workflow
    return building_list_uri
```

TOTAL FRACTION

```
import sys
from SPARQLWrapper import SPARQLWrapper, JSON, POST

# Define SPARQL endpoints as constants
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT                                                    =
"http://localhost:7200/repositories/DisaggregationProbass9/statements"
INSERT_CHUNK_SIZE = 500  # Number of buildings to insert per batch
```

```
def unpack_rdfs_list(list_head_uri, sparql_endpoint):
    """
    Takes the URI of the head of an rdfs:List and queries the graph
    to return a standard Python list of its members' URIs.
    """
    if list_head_uri == "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil":
        return []

    sparql = SPARQLWrapper(sparql_endpoint)
    sparql.setReturnFormat(JSON)
    query = f"""
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    SELECT ?member WHERE {{ <{list_head_uri}> rdf:rest* / rdf:first ?member . }}
    """
    sparql.setQuery(query)
    results = sparql.query().convert()
    if not results["results"]["bindings"]:
        return []
    return [item["member"]["value"] for item in results["results"]["bindings"]]


def doTotalFraction(building_list_uri):
    """
    Calculates the total fraction for each building based on its
    area ratio and density ratio.
    """
    # --- Step 0: Unpack the list of buildings ---
    print("Step 0: Unpacking the rdfs:List...")
    building_uris = unpack_rdfs_list(building_list_uri, SPARQL_QUERY_ENDPOINT)
    if not building_uris:
        print("  -> Input list is empty or invalid. Nothing to process.")
        return building_list_uri

    print(f"  -> Found {len(building_uris)} buildings in the list.")

    sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
    sparql.setReturnFormat(JSON)
    sparql.setMethod(POST)

    # --- Step 1: Fetch all area and density ratios in a single query ---
    print("Step 1: Fetching all area and density ratios in a single batch query...")
    values_clause = " ".join([f"<{uri}>" for uri in building_uris])
```

```
    ratios_query = f"""
    PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
    SELECT ?building ?areaRatio ?densityRatio
    WHERE {{
        VALUES ?building {{ {values_clause} }}
        ?building bdasy:hasAreaRatio ?areaRatio ;
                  bdasy:hasDensityRatio ?densityRatio .
    }}
    """
    sparql.setQuery(ratios_query)
    results = sparql.query().convert()["results"]["bindings"]

    # --- Step 2: Calculate products and the sum of products in memory ---
    print("Step 2: Calculating products and sum in memory...")
    building_products = []
    total_product_sum = 0.0
    for row in results:
        area_ratio = float(row['areaRatio']['value'])
        density_ratio = float(row['densityRatio']['value'])
        product = area_ratio * density_ratio

        building_products.append({'uri': row['building']['value'], 'product': product})
        total_product_sum += product

    if total_product_sum == 0:
        print("  -> Sum of products is zero. Cannot calculate fractions. Aborting.")
        return building_list_uri

    print(f"  -> Sum of (AreaRatio * DensityRatio) products: {total_product_sum}")

    # --- Step 3: Insert the new total fractions in chunks ---
    print(f"Step 3: Preparing to insert total fractions in chunks of {INSERT_CHUNK_SIZE}
buildings...")
    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)

    for i in range(0, len(building_products), INSERT_CHUNK_SIZE):
        chunk = building_products[i:i + INSERT_CHUNK_SIZE]
        print(f"  -> Processing chunk {i // INSERT_CHUNK_SIZE + 1}...")

        insert_triples = []
        for building in chunk:
            # Calculate the final fraction
```

```
            total_fraction = building['product'] / total_product_sum

            fraction_triple = f'<{building["uri"]}>
<http://www.example.com/ontology/buildingareadasy#hasTotalFraction>
"{total_fraction}"^^<http://www.w3.org/2001/XMLSchema#double> .'
            insert_triples.append(fraction_triple)

        insert_query_body = "\n".join(insert_triples)
        batch_insert_query = f"INSERT DATA {{ {insert_query_body} }}"

        update_sparql.setQuery(batch_insert_query)
        update_sparql.query()

    print("  -> All chunks inserted successfully.")

    # Return the original list URI for the next step in the workflow
    return building_list_uri
```

DISAGGREGATION

```
import sys
from SPARQLWrapper import SPARQLWrapper, JSON, POST

# Define SPARQL endpoints as constants
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT =
"http://localhost:7200/repositories/DisaggregationProba9/statements"
INSERT_CHUNK_SIZE = 500  # Number of buildings to insert per batch

def unpack_rdfs_list(list_head_uri, sparql_endpoint):
    """
    Takes the URI of the head of an rdfs:List and queries the graph to return a standard Python
list of its members' URIs.
    """
    if list_head_uri == "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil":
        return []

    sparql = SPARQLWrapper(sparql_endpoint)
    sparql.setReturnFormat(JSON)
    query = f"""
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
    SELECT ?member WHERE {{ <{list_head_uri}> rdf:rest* / rdf:first ?member . }}
    """
    sparql.setQuery(query)
    results = sparql.query().convert()
    if not results["results"]["bindings"]:
        return []
    return [item["member"]["value"] for item in results["results"]["bindings"]]


def doDisaggregation(source_zone_uri, building_list_uri):
    """
    Calculates the disaggregated population for each building in a list
    by multiplying the total adjusted population by each building's total fraction.
    """
    # --- Step 0: Unpack the list of buildings ---
    print("Step 0: Unpacking the building rdfs:List...")
    building_uris = unpack_rdfs_list(building_list_uri, SPARQL_QUERY_ENDPOINT)
    if not building_uris:
        print("  -> Input list is empty or invalid. Nothing to process.")
        return building_list_uri

    print(f"  -> Found {len(building_uris)} buildings in the list.")

    sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
    sparql.setReturnFormat(JSON)
    sparql.setMethod(POST)

    # --- Step 1: Get the adjusted population from the source zone ---
    print(f"Step 1: Fetching adjusted population from source zone <{source_zone_uri}>...")
    query_pop = f"""
    PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
    SELECT ?adjustedPop WHERE {{
        <{source_zone_uri}> bdasy:hasAdjustedPopulation ?adjustedPop .
    }}
    """
    sparql.setQuery(query_pop)
    results_pop = sparql.query().convert()["results"]["bindings"]
    if not results_pop:
        raise ValueError(f"No adjusted population found for sourceZone {source_zone_uri}")
    adjusted_population = float(results_pop[0]["adjustedPop"]["value"])
    print(f"  -> Adjusted population found: {adjusted_population}")

    # --- Step 2: Fetch all total fractions in a single query ---
```

```
    print("Step 2: Fetching all total fractions in a single batch query...")
    values_clause = " ".join([f"<{uri}>" for uri in building_uris])

    fractions_query = f"""
    PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
    SELECT ?building ?totalFraction
    WHERE {{
        VALUES ?building {{ {values_clause} }}
        ?building bdasy:hasTotalFraction ?totalFraction .
    }}
    """
    sparql.setQuery(fractions_query)
    fraction_results = sparql.query().convert()["results"]["bindings"]

    # --- Step 3: Calculate disaggregated population and prepare for insert ---
    print("Step 3: Calculating disaggregated population and preparing for batch insert...")
    insert_triples = []
    for row in fraction_results:
        building_uri = row['building']['value']
        total_fraction = float(row['totalFraction']['value'])

        # Calculate the final disaggregated population for this building
        disagg_pop = adjusted_population * total_fraction

        # Create the triple string for this building
        pop_triple = f'<{building_uri}>
<http://www.example.com/ontology/buildingareadasy#hasDisaggregatedPopulation>
"{disagg_pop}"^^<http://www.w3.org/2001/XMLSchema#double> .'
        insert_triples.append(pop_triple)

    # --- Step 4: Insert the new disaggregated populations in chunks ---
    print(f"Step 4: Preparing to insert disaggregated populations in chunks of
{INSERT_CHUNK_SIZE} buildings...")
    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)

    for i in range(0, len(insert_triples), INSERT_CHUNK_SIZE):
        chunk = insert_triples[i:i + INSERT_CHUNK_SIZE]
        print(f"  -> Processing chunk {i // INSERT_CHUNK_SIZE + 1}...")

        insert_query_body = "\n".join(chunk)
        batch_insert_query = f"INSERT DATA {{ {insert_query_body} }}"
```

```
        update_sparql.setQuery(batch_insert_query)
        update_sparql.query()

    print("  -> All chunks inserted successfully.")

    # Return the original list URI for the final aggregation step
    return building_list_uri
```

ARBITRARY AGGREGATION

```
import sys
from SPARQLWrapper import SPARQLWrapper, JSON, POST

# Define SPARQL endpoints as constants
SPARQL_QUERY_ENDPOINT = "http://localhost:7200/repositories/DisaggregationProba9"
SPARQL_UPDATE_ENDPOINT =
"http://localhost:7200/repositories/DisaggregationProba9/statements"

def unpack_rdfs_list(list_head_uri, sparql_endpoint):
    if list_head_uri == "http://www.w3.org/1999/02/22-rdf-syntax-ns#nil": return []
    sparql = SPARQLWrapper(sparql_endpoint)
    sparql.setReturnFormat(JSON)
    query = f"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> SELECT
?member WHERE {{ <{list_head_uri}> rdf:rest* / rdf:first ?member . }}"
    sparql.setQuery(query)
    results = sparql.query().convert()["results"]["bindings"]
    return [item["member"]["value"] for item in results] if results else []


def doArbitraryAggregation(building_list_uri, arbitrary_unit_query):
    print("Step 0: Unpacking the rdfs:List...")
    building_uris = unpack_rdfs_list(building_list_uri, SPARQL_QUERY_ENDPOINT)
    if not building_uris:
        print("  -> Input list is empty or invalid. Nothing to aggregate.")
        return "http://www.example.com/data/areainterest/EmptyResultTargetZone"
    print(f"  -> Found {len(building_uris)} buildings to aggregate.")

    sparql = SPARQLWrapper(SPARQL_QUERY_ENDPOINT)
    sparql.setReturnFormat(JSON)
    sparql.setMethod(POST)
```

```
    # First, get the target zone URI from its query
    sparql.setQuery(arbitrary_unit_query)
    target_zone_results = sparql.query().convert()["results"]["bindings"]
    if not target_zone_results:
        raise ValueError("Arbitrary unit query returned no results.")
    target_zone_uri = target_zone_results[0]['arbitraryUnit']['value']

    print(f"  -> Found Target Zone: <{target_zone_uri}>. Now fetching its geometry...")
    get_geom_query = f"""
    PREFIX geo: <http://www.opengis.net/ont/geosparql#>
    SELECT ?wkt
    WHERE {{
        # First hop: from the Area of Interest to its Geometry Instance
        <{target_zone_uri}> geo:hasGeometry ?geom .
        # Second hop: from the Geometry Instance to the WKT literal
        ?geom geo:asWkt ?wkt .
    }}
    """
    sparql.setQuery(get_geom_query)
    geom_results = sparql.query().convert()["results"]["bindings"]
    if not geom_results:
        raise ValueError(
            f"Could not find geometry for target zone <{target_zone_uri}>. Check that the 'geo:' prefix
in your data file matches 'http://www.opengis.net/ont/geosparql#'.")

    target_wkt = geom_results[0]['wkt']['value']

    print("Step 1 & 2: Spatially filtering buildings and summing population in SPARQL...")
    values_clause = " ".join([f"<{uri}>" for uri in building_uris])

    aggregation_query = f"""
    PREFIX geo: <http://www.opengis.net/ont/geosparql#>
    PREFIX geof: <http://www.opengis.net/def/function/geosparql/>
    PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
    SELECT (SUM(?pop) as ?totalPopulation)
    WHERE {{
        VALUES ?building {{ {values_clause} }}
        ?building geo:hasGeometry ?buildingGeom ;
                bdasy:hasDisaggregatedPopulation ?pop .
        ?buildingGeom geo:asWKT ?buildingWKT .
        BIND("{target_wkt}"^^geo:wktLiteral AS ?targetGeom) .
        FILTER(geof:sfWithin(?buildingWKT, ?targetGeom))
    }}
```

```
    """
    sparql.setQuery(aggregation_query)
    results = sparql.query().convert()["results"]["bindings"]

    total_population = 0.0
    if results and results[0].get("totalPopulation"):
        total_population = float(results[0]["totalPopulation"]["value"])

    print(f"  -> Aggregated population in target zone: {total_population}")

    update_sparql = SPARQLWrapper(SPARQL_UPDATE_ENDPOINT)
    update_sparql.setMethod(POST)
    insert_query = f"""
    PREFIX bdasy: <http://www.example.com/ontology/buildingareadasy#>
    INSERT DATA {{ <{target_zone_uri}> bdasy:hasAggregatedPopulation
"{total_population}"^^xsd:double . }}
    """
    update_sparql.setQuery(insert_query)
    update_sparql.query()

    print("  -> Final data inserted successfully.")

    return target_zone_uri
```

CURRICULUM VITAE

Karlo Kević, born in 1994 in Split, Croatia graduated the Technical School of Construction and Geodesy and gained a qualification of Land Surveying Technician in 2013. After finishing bachelor studies at Faculty of Civil Engineering, Architecture and Geodesy at University of Split in 2016, he obtained master's degree in geodesy and geoinformatics at Faculty of Geodesy, University of Zagreb in 2019. Following upon his studies, he started his professional career by enrolling in doctoral study at Faculty of Geodesy in 2019 while working as teaching and research assistant at Chair of Geoinformation. In the scope of his work position, he participates in several teaching courses at bachelor and master level and at the same time was actively involved in Erasmus and Horizon scientific projects as an early-stage researcher.

Throughout his professional career to date, he participated and presented at several national and international scientific conferences. Adjacent to that he is author of several scientific papers published in international journals and conference proceedings.