A MODEL-DRIVEN APPROACH FOR DEVELOPING REST-BASED GEOSPATIAL WEB APPLICATION USING UML PROFILES

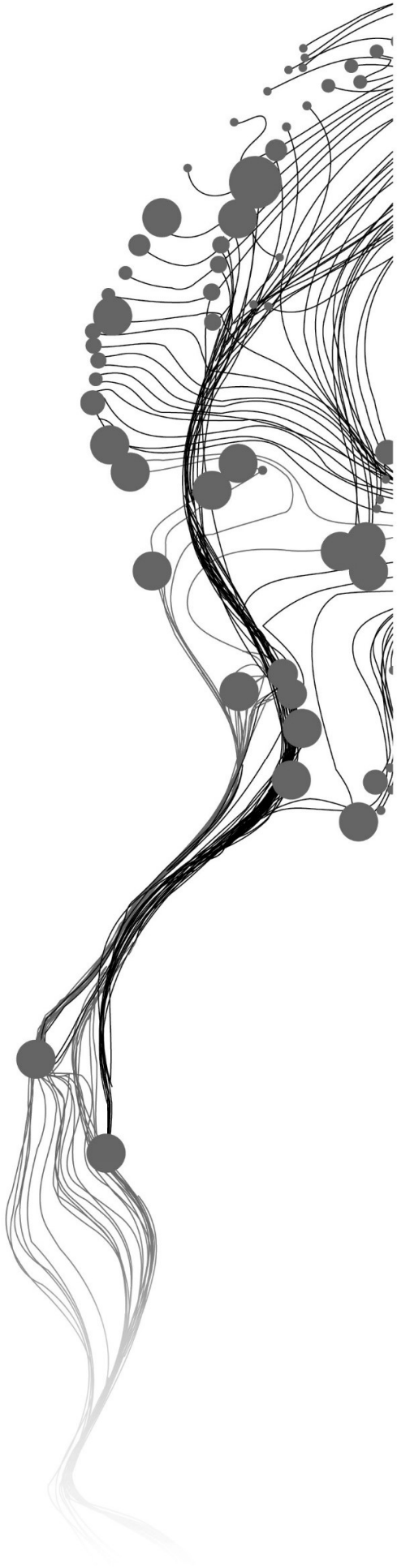
RIFQI ALFADHILLAH SENTOSA

JULY, 2024

SUPERVISORS:

dr. J.M. Morales Guarin

dr.ir. R.A. de By



A MODEL-DRIVEN APPROACH FOR DEVELOPING REST-BASED GEOSPATIAL WEB APPLICATION USING UML PROFILES

RIFQI ALFADHILLAH SENTOSA

Enschede, The Netherlands, July, 2024

Thesis submitted to the Faculty of Geo-Information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

SUPERVISORS:
dr. J.M. Morales Guarin
dr.ir. R.A. de By

THESIS ASSESSMENT BOARD:
dr. F.O. Ostermann
Prof.dr.ir. P.J.M. van Oosterom

ABSTRACT

The development of geospatial web applications (GWAs) has seen significant advancements, one of which is with Model-Driven Development (MDD) and Model-Driven Architecture (MDA). MDA provides a systematic transformation of high-level models into executable computer code of applications, which provides a significant value as a development methodology for web application.

This study focuses on leveraging the development process of REST-based GWAs using Unified Modelling Language (UML) profiles and used it within MDA framework. The research aims to streamline GWA development, enabling users with inadequate level of web application development skills to a running REST-based GWA for their use case. However, the current state-of-the-art in GWA development lacks a standardized, reproducible approach that integrates UML profiles with MDA for REST-based architectural style. This gap limits the accessibility and scalability of GWA development for non-expert users.

This thesis proposed a new development approach that integrates UML profiles into the MDA framework for developing REST-based GWAs. This study involved collecting common functions from existing GWAs, analysing their functional requirements, and developing corresponding UML profiles. These profiles were then used to create Platform Independent Models (PIMs) and Platform Specific Models (PSMs), which were transformed into executable code.

The methodology was demonstrated through a detailed implementation process, resulting in a reproducible approach for developing GWAs. Several findings include the successful creation of UML profiles that capture common GWA functionalities and the development of efficient model transformation rules using the proposed approach.

The proposed approach lowers the barrier for GWA development, enabling GIS/RS practitioners and developers to create customized web applications with minimal coding effort. This approach enhances the scalability and accessibility of GWA development, potentially leading to broader adoption and innovation in the field of REST-base geospatial web applications.

Keywords: Geospatial Web Application, Model Driven Development, Model Driven Architecture, UML, REST API, OGC API Standards

ACKNOWLEDGEMENTS

Everything has come to an end. The exciting idea, the enjoyable learning curve, and the full-pressure finishing that are this thesis research topic – they must end too. Credits are given to where they are due.

Biggest praises be to Allah SWT., the One and Only, who let everything came and also ended accordingly.

I would like to thank dr. J. M. Morales Guarin for introducing me to this research topic and dr.ir. R.A. de By for the provided insights. I would also convey a big gratitude to drs. Barend Köbben as the procedural advisor who always keep me in the right direction with his timely supports and encouragements.

I am very grateful for LPDP scholarship who made my abroad study dream became reality.

My parents and siblings. The support system who always reminds me whom I fight for.

ITC Indonesian friends. Clava, Wibi, Nasir, Andi, Ghaly, Salsa, Sry, Ganda. All shared fun, potlucks, trips, jokes, stickers, hardships, supports, and motivations are forever unforgettable.

ITC international friends. Amy, Amin, Gamil, Belise, Maria, and the rest of the batch whom I cannot say one by one without turning this acknowledgement into a student database. Thank you for making the best atmospheres and memories to a beautiful journey of a Master study.

PPIE team. An amazing group of people who let me grow in the organization context, but also kind enough to spare me the chance to focus when I needed it.

My life partner Zahra. Si Biu. The most patient and kind girl I have ever loved. Thank you for believing in me and always reminding me to believe in myself.

Thank you for everyone helping me in this study.

Finally, the last words below are for my future self.

Years from the moment this passage was written.

You sacrificed a lot to reach until this point.

All the important people you left behind.

All the chance to continue your career.

All the failures and lost opportunities.

Now that everything comes to an end.

Were the sacrifices worth something?

Your journey for proofs began.

I hope you find them.

On a train from Utrecht to Enschede, 1 July 2024

Rifqi Alfadhillah Sentosa

Table of Contents

1. RESEARCH INTRODUCTION

1.1. Introduction

A Web application is software that depends on or uses the World Wide Web infrastructure to execute properly (Gellersen & Gaedke, 1999). A sophistication in this is geospatial functionality that leverages geospatial information, thanks to global access of location-based information and services (Veenendaal, 2015). A supportive factor in its realization is the recently updated OGC API standard by Open Geospatial Consortium (OGC, 2023) which advocates the use of resource-oriented REST architecture in geospatial web applications (GWA). It aligns with the concept of REST API, one of the most accepted architectural styles in the web. It allows web application specifications to be broken down into independent components, improving the application's modularity, unlocking future interoperability, and the possibility to use diverse data formats in one common architectural style (Blanc et al., 2022).

Development of geospatial functionality in a web application is not an easy task for GIS and Remote Sensing (RS) practitioners, who are often unfamiliar with web development methods. Software development as a field of engineering, including web application development, is technically complex and requires advanced design skills to develop the application (Barry & Lang, 2003). Traditional software development life cycle (SDLC) concepts like requirement analysis, wireframing, and testing, including methods like agile development and the waterfall model, are a field by itself and requires skills that GIS/RS practitioners often are unfamiliar with.

Various high-level abstraction techniques aim to bridge between domain-based knowledge and software development such as Model-Driven Architecture (MDA) (OMG, 2014), web-based engineering, including UML-based Web Engineering (UWE), Model-Driven Web Engineering (MDWE) and even agile-based Mockup-Driven Development (MockupDD) (Rivero et al., 2012). They all provide possibilities to develop applications, but they come with a steep learning curve for potential non-developer users. Fortunately, the sophistications in implementation of REST-based GWAs and model-driven development techniques allow the development for an easier approach for non-developer users to create their own GWA. To do this, high-level conceptual designs that contain components in GWA and the documented approach to use it in developing REST-based GWAs are necessary.

This research aims to develop UML profiles of REST-based GWA components and to document a model-driven development of REST-based GWA that leverages those profiles. Model Driven Architecture (MDA) was chosen as the model-driven approach, allowing the use of UML profiles to solve the complexities of development using multiple level of abstraction models (OMG, 2014). MDA initiative is a well-researched subject, with examples including Elleuch et al. (2007) and Jovanović et al. (2013) who brought Model Driven Engineering, a concept related to the OMG's MDA initiative, to software architecture and user interface domain; Betari et al. (2018) attempted and modelled the transformation of Computation Independent Model (CIM), the highest level abstraction model in MDA principle; and Meliá et al. (2005) who approaches architecture modelling with MDA's lingua franca of Unified Modelling Language (UML).

The research started by collecting various types of geospatial web applications and performing domain modelling from the perspective of REST architectural style and OGC API standards to retrieve the GWA functional requirements. These requirements were translated into UML stereotypes and were packaged

into UML profiles. The developed UML profiles were then applied into an MDA implementation of REST-based GWA development, where the profiles were used to create an example of a user-defined Platform Independent Model (PIM) then were transformed down to Platform Specific Models (PSM), which results in computer code of a running GWA. This implementation results in a documented approach that makes the demonstrated approach reproducible by other users.

The expected results of this research project are the UML profiles containing stereotypes of common REST-based GWA components and a documented MDA-based GWA development. The results are expected to improve state-of-the-art in several ways that benefit multiple types of users. The UML profiles and the documented MDA-based development become the initial attempt to develop a new GWA development approach. This approach allows non-developer users like most GIS/RS practitioners to design their own GWA by only focusing on the higher-level design of what they want to include in their GWA. The UML profiles and the documented approach are also valuable for developers to continue developing the approach, which open to more technology stacks and widen the possibilities of the types of GWAs that can be developed. In this way, users can seamlessly integrate geospatial functionality in their web applications with less time, effort, and even code lines.

1.1.1. Research Objectives

The main objective of this research is to develop an MDA-based approach for REST-based GWA development based on UML profiles. Based on the main objective, there are four sub-objectives to this research:

1. To conceptualize the functional requirements of common REST-based GWA functionalities from multiple GWA observations.
2. To develop UML profiles from the functional requirements of REST-based GWA and the model transformations for MDA approach.
3. To demonstrate how the proposed UML profiles and the model transformations are being used in developing a REST-based GWA.
4. To evaluate the proposed approach and its resulting REST-based GWA.

1.1.2. Research Questions

Related to the first sub-objective:

1. What are the common functions found from the GWA observations?
2. What are the observed functional requirements from the selected GWA functionalities?
3. What are the potential class, attribute, operation, and relationship semantics found from the functional requirements?

Related to the second sub-objective:

1. What are the proposed UML stereotypes for developing Platform Independent Model (PIM)?
2. What are the proposed UML stereotypes for developing Platform Specific Model (PSM)?
3. How do the proposed UML stereotypes be developed into UML profiles?

Related to the third sub-objective:

1. How do the proposed UML profiles support the development of the Platform Independent Model (PIM)?
2. What is the model transformation rules to transform the PIM to Platform Specific Models (PSM)?
3. What is the model transformation rules to generate the GWA computer code from the PSM?

Related to the fourth sub-objective:

1. What are the strengths of the proposed approach and its resulting GWA?

2. What are the weaknesses and limitations of the proposed approach and its resulting GWA?
3. What are the future opportunities and recommendations for the proposed approach?
4. What are the future challenges and threats for the proposed approach?

1.1.3. Research Outline

The main phases of this research are the literature review, the methodology of the research, the execution of the research methodology; the implementation step; the evaluation of the research; then lastly the conclusion. Below is the detailed structure of this thesis research:

Chapter 1 is the introduction. This chapter provides the fundamental information of the research, including the research background, objectives, and questions.

Chapter 2 is the literature review. This chapter gives the basis of the research. First, the concept of web application development is introduced. It includes Model-Driven Development which overarch the Model-Driven Architecture (MDA). MDA as the main theme of this research is elaborated in detail. Afterwards, the concepts of REST architectural style and OGC API standards are introduced.

Chapter 3 is the research method. This chapter introduces the steps that are performed in this research. The steps include domain modelling, UML profiling, model transformation development, MDA implementation, and evaluation.

Chapter 4 is the domain modelling steps. This chapter answers the sub-objective 1 and its research questions. The GWA observations and their results are explained in this chapter. Afterwards, the analyses to retrieve functional requirements and the potential semantics for UML elements such as classes, attributes, operations, and relationships are explained.

Chapter 5 is the UML profile and model transformation development. This chapter answers the sub-objective 2 and its research questions. The development of UML profiles from the result of previous chapter are explained. Then, the development of model transformations for MDA approach are explained, including PIM-to-PSM and PSM-to-code transformation.

Chapter 6 is the MDA approach implementation. This chapter answers the sub-objective 3 and its research questions. The developed UML profiles and the model transformation approach are performed to generate a GWA example.

Chapter 7 is the result evaluation. This chapter answer the sub-objective 4 and its research questions. The implementation of the developed approach and the leveraged UML profiles are evaluated based on SWOT framework. The strengths, weakness, future opportunities, and threats are elaborated by observing the demonstrated approach.

Chapter 8 is the conclusion of the research. This chapter summarizes the answers to each research questions. This chapter also includes limitations and recommendations of the thesis research for future considerations.

2. LITERATURE REVIEW

This chapter provides an overview of the main concepts of the research. Firstly, the concept of web application development that is introduced. It includes the role of Model-Driven Development (MDD). Then, the Model Driven Architecture (MDA) is explained as part of Model-Driven Development. The detailed framework about MDA is elaborated which includes the use of Unified Modelling Language (UML), the MDA level of abstractions, and model transformations. The literature review finishes with the 2 subjects of geospatial web applications where REST architecture and OGC API standards are introduced.

2.1. Web Application Development

Web application is a software system consisting of web pages and database where users can connect via their browser through a network (Y. F. Li et al, 2014). A web application involves the frontend for the web page visualizations and user interactions and the backend for database and data processing (Pop & Altar, 2014). The exponential sophistications in web applications have brought many benefits in the current world as they unlocked global accesses to many aspects of life. Abundance of new technologies in developing a web application are released in rapid pace over the years. Therefore, it is important for new development methodologies are pursued to keep up with the innovations (Molina-Rios & Pedreira-Souto, 2020).

The development of web applications has been closely mirroring those of software development methodologies. Web Development Life Cycle (WDLC) (Kamatchi et al., 2013) follows Software Development Life Cycle (SDLC) techniques to in laying down the typical steps that developer teams go through in developing a web application: identifying website objectives, gathering requirements, perform the design process, test the application through multiple levels, deploy, and do maintenance.

2.1.1. Current Web Development Methodologies

In current industry, several web application development methodologies have been implemented. Waterfall model was introduced by Royce (1970) as sequence-based development process to follow through like a consecutive list of tasks to be performed to build a software, allowing straightforward ways in collecting requirement and managing team resources at the cost of low flexibility and limited end-user involvement.

Agile methodology, a set of iterative and incremental approaches with high adaptability, involves sprints where short iterations are implemented to rapidly deploy parts of software to be tested. The family of Agile approaches such as Scrum and Kanban give the advantages of flexibility, collaboration, and iteration of developing software and web applications while facing challenges in documentation, high dependency to customer collaboration, and requirements for experienced developers to implement (Hossain, 2023). Another example is Model Driven Development (MDD) that uses abstractions from capturing requirements of the application until creating the application itself.

2.1.2. Model Driven Development (MDD)

Model Driven Development (MDD) leverages the concept of using models in representing applications in different levels of abstraction, and using commonly automatic, top-down model transformations to define the specificity of the previous model to other models, become ultimately deliver the final application

(Rivero et al., 2012). This approach is known by several terminologies laid down by Whittle et al. (2014) including Model Driven Engineering (MDE) as the superset of MDD and Model Driven Architecture (MDA) as one of MDD instances that comply with OMG standards. Despite the interchangeable semantics, the general model driven concept is the same, and it has been proven to be beneficial to software and application development. Model Driven Development is beneficial in development process as it is capable to document software architecture and eventually perform code generation, especially in domain-based companies and projects where new developments can conveniently be performed thanks to the abstraction modelling MDD proposes (Whittle et al., 2014).

Multiple software and web engineering methods apply MDD principles. Distante et al., (2007) classified this kind of web application design methods into two categories: the conceptual design-focused ones that prioritize on showing what is required for the application to perform, where approaches such as Ubiquitous Web Application (UWA) and Object-Oriented Web Solutions (OOWS) are fell into; and the logical design-focused that prioritize on showing how to implement those requirements, where a proposition from Conallen (1999) on building web application with UML is fell into. Anything in between is considered hybrid methodologies that covers both logical and conceptual designs, which includes popular methods like UML-based Web Engineering (UWE) and Web Modeling Language (WebML).

2.2. Model Driven Architecture (MDA)

The Model Driven Architecture (MDA) is a software designing approach to derive value and to provide guidelines for designing specifications of a software system using models (OMG, 2014). The MDA is a well-documented family of standards maintained by Object Management Group (OMG) that has been a centre of research and being implemented in various domains such as IT, healthcare, manufacturing, and government.

2.2.1. MDA Abstractions

The MDA promotes abstraction of complex details using high-level models to provide a structured representation of a, making it easier for developers to build and understand a complex system. The MDA promotes the use of transformable models that ultimately result in executable codes (Xiao & Greer, 2009). The main mechanism of MDA starts with conceptualize the high-level requirements, then specify the platform-agnostic solutions, before finally adopts the solutions using selected technologies and platforms.

The MDA approach is based on three different viewpoints: Computer Independent Model (CIM) for expressing the working logic of an application, Platform Independent Model (PIM) for describing the information system without being tied to any platform, and lastly Platform Specific Model (PSM) for modelling the information system with more specificity using certain platforms. The models be transformed from one level to another using defined transformation rules, with each level of transformation promotes various amount of automation.

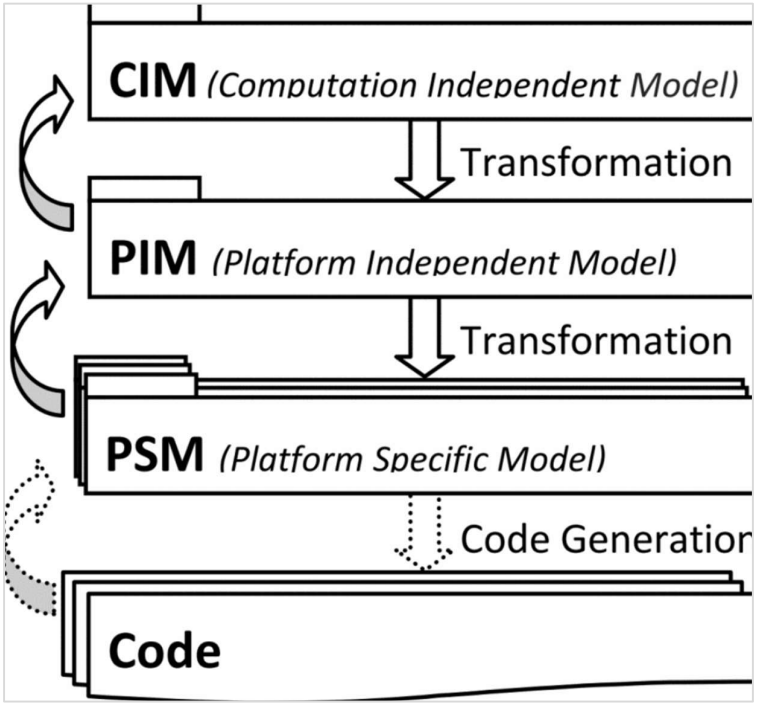


Figure 2. 1 The abstraction models as core concepts of Model Driven Architecture (Kriouile, 2014)

Computation Independent Models (CIM) describe systems from a high-level perspective, focusing on the environment and requirements without technical details. It is commonly being referred as business model and domain model. Compared to the other abstraction models in MDA, CIM is seen as model used for analysis that helps to meet business and user requirements (Singh & Sood, 2010).

Platform Independent Models (PIM) emphasize the creation of models in which the specifications and functionalities of the models are separated from implementations of any specific platforms. This level of abstraction disregards any kind of operating systems and programming languages (Betari et al., 2018). The use of PIM promotes technical flexibility and easier adaptation to different technologies in later abstraction (Ameller et al., 2015). PIM is seen as mechanism to describe the concepts of a domain (Singh & Sood, 2010).

Platform Specific Models (PSM) brings PIM to adopt a particular technological context. PSM details the implementation of a system including specific technologies, platforms, and frameworks, bridging the gap between the design and the actual code. PSM is seen as model used for implementation (Singh & Sood, 2010).

Model transformation is important part of the model-driven approach. By using selected model transformation, a model can be automatically transformed into another model (Thomas, 2004). Model transformations are the mechanism to derive PSM models from PIMs. Model transformations in MDA typically use OMG-standardized transformation language such as Design Specific Language, (DSL), Query/View/Transformation (QVT), and Unified Modelling Language (UML).

2.2.2. Unified Modelling Language (UML)

UML is one of modelling languages developed by Object Management Group (OMG) that focuses on the use of graphical models to describe properties of a software system (Duke et al., 1997). It is considered as

lingua franca in software engineering (Wimmer et al., 2006). UML provides solutions based on specific user goals, such as UML class diagram, use case diagram, and activity diagram (Shcherban et al., 2021).

UML is a widely accepted modelling language by the software and web application development industry. UML having a lot of benefits such as the clear visual abstractions of UML and its widespread adoption. UML is also developed and standardized by OMG, the same institution that developed Model Driven Architecture (MDA) approach which adds UML dependability factor as a modelling language for the approach.

There are two types of UML diagram: structural and behavioural. Structural diagrams, such as class diagram, deployment diagram, profile diagram, and component diagram (Ma & Chao, 2020), model elements in a domain to be static at a specific point in time (OMG, 2017). Behavioural diagrams, such as use case diagram, activity diagram, and sequence diagram (Ma & Chao, 2020), model domain elements as they change over time (OMG, 2017). In this research, class diagram and profile diagram are heavily used alongside with the concept of UML stereotype that relates to the latter.

2.2.2.1. UML Class Diagram

A UML class diagram is a structural diagram that helps in designing and illustrating classes in a system and its internal relationships (Gosala et al., 2021). UML class diagram is commonly and heavily used to model software components in a development, especially with its close relation with Object-Oriented paradigm. A class diagram typically consists of attributes (data member of the class), operations (behaviors or functions the class can perform), and relationships which are designed to represent elements of a system or domain.

2.2.2.2. UML Profile Diagram

UML profile diagram is a structural diagram with an extension mechanism to cater UML metamodel for specific domain case using set of stereotypes, tagged values, and constraints for their UML extension (Giubergia et al., 2014). A metamodel refers to a predefined language, structure of objects, and concepts for defining a model. To extend the UML metamodel, UML profiles leverage the use of stereotypes, indicated as stereotype in the profile diagram and has the same behaviour as a UML class. Using stereotypes and other capabilities namely tagged values and constraints, this UML extension mechanism allows the existing UML meta classes from existing metamodels to be used in modelling domain-based problems without changing the original UML vocabularies (UML, Meta Meta Models and Profiles - Metamodeling, MOF, UML, Profiles, Etc., n.d.).

2.3. REST Architectural Style

REST (Representational State Transfer) is an architectural style that design applications where resources are identified by URLs and manipulated through standard HTTP methods. REST architectural style was first proposed by Roy Fielding in his dissertation (Fielding et al., 2017) where REST was defined as an architectural style derived from the aggregation of several architectural style with web constraints, resulting in a style that are suitable for network-based applications. REST architectural style is a widely adopted approach for remote programming resources to benefit from remote APIs or services (Rodríguez et al., 2016).

REST-based design promotes scalability and interoperability of web services and APIs (Rodríguez et al., 2016). Applications that adhere to REST architectural style abide to the following REST core principles:

- Resource addressability: Resources should be uniquely identified using Uniform Resource Identifier (URI).
- Resource representation: The internal structure and state of resources are kept hidden from the clients/users as they are represented only by the resource representations, XML and JSON format being the common formats, in help of HTTP protocol to manage the representations.
- Uniform interface: Resources are handled using standardized methods of HTTP protocol.
- Statelessness: Client request to the API should contain all information required to be processed, so that the server does not keep any interaction state.
- Hypermedia as the engine of state: Links between resources allow users to understand the relationships and the interaction state.

2.4. OGC API Standards

OGC API standards is the latest geospatial set of standards introduced in 2016 by Open Geospatial Consortium (OGC) that allows modular development for geospatial services and systems using resource-based oriented principle (OGC, n.d.-c). OGC API standards define the consistent and modular building block for spatial Web APIs (*OGC API - Common - Overview*, n.d.).

OGC API standards is the latest initiative from OGC to facilitate the adoption of the latest OpenAPI Specifications 3.0 (OAS), a programming language-agnostic described standard for HTTP API, the protocol that bases REST architectural style. Before this, the consortium was focused on maintaining OGC Web Services (OWS) since it was designed in the early 2000. It includes older version of geospatial standards such as Web Map Service (WMS) and Web Feature Service (WFS) that implement action-based Remote Procedure Call (RPC) architectural style which was indeed the state of the art during its time (OGC, n.d.-c).

OGC API standards adhere REST architectural style principles which made the new standards more lightweight and flexible than its predecessor standards. OGC API standards is OGC attempt to follow current trend in web application development and the enforcement of its family of standards toward geospatial web application will be beneficial in terms of future interoperability with modern tools, especially in promoting geospatial data integration in modern web technologies including web applications (Blanc et al., 2022).

Based on its status, the standards can be categorized into two categories: Approved Standards and the ones that are not yet approved. According to OGC API official website page (OGC, n.d.-b), the currently OGC API Approved Standards as of June 2024 are as follow:

- OGC API – Features
  The standard for handling "things in the real world that are of interest" (OGC, n.d.-e)
- OGC API – Common
  The standard for implementing Web API that confirms to OGC API Standards (OGC, n.d.-c)
- OGC API – Tiles
  The standard for implementing Web API that support the retrieval of several types of tiled geospatial information, such as map tiles and vector tiles (OGC, n.d.-i)
- OGC API – Processes
  The standard for executable processes a client can call via Web API (OGC, n.d.-g)
- OGC API – EDR
  The standard that provides interfaces to access spatiotemporal data resource and perform spatial queries (OGC, n.d.-d)

Several mentionable OGC API Standards that has not yet gotten approval status are as follow:

- OGC API – Maps
  The standard for requesting maps through Web API (OGC, n.d.-f)
- OGC API – Records
  The standard for handling metadata of resources (OGC, n.d.-h)

3. RESEARCH METHOD

This chapter explains the overview of the methodology used in the research. The important part of this research is to develop a methodology of MDA implementation for developing REST-based geospatial web applications, which includes providing resources and documentation to ensure the developed methodology is reproducible. Figure 5.1 shows the flowchart of the research methods.

First, the common functions of multiple chosen GWAs were collected by observing all explored features and components in their interfaces. Next, the collected common functions were categorized based on its similarities in the purpose and the possible logic behind it. Then, some common functions with different purpose and web app logics were analysed and described using natural language for their functional requirements. These functional requirements contained possible semantics for classes, attributes, operations, and relationships for developing the UML profile.

Afterwards, the developments of UML profile diagrams purposed for building PIM and PSM were performed by addressing each observed potential semantics from the abstracted functional requirements. Then, the transformation rules for model transformation were developed for both PIM-to-PSM transformation and code generation.

After the abstraction models and the model transformations were laid out, a implementation on how to implement the MDA approach using the developed UML profiles and the model transformations was performed. The implementation included building an instance of PIM, transforming it into PSM using the transformation rules, and generating computer codes of GWA by transforming the PSM.

All the mentioned steps above were documented along with the detailed consideration and justification as to why the step are performed in certain way. Most of the analyses and design steps were performed using the designing application Figma for its designing versatility that also include UML modelling. The model transformation used Visual Studio Code for manifesting the transformation into script.

This research caters to three different types of users based on the level of development skills and the purpose of implementing this research:

User Type #1: The Designer

This user type includes those who are interested in building GWA but have no adequate web application development skills and knowledge. For this user type, the proposed approach provides UML profiles so they can design a GWA using their knowledge of the domain. The expectation is that this user type manages to express their design using the proposed UML profiles and then realise it using the transformation steps.

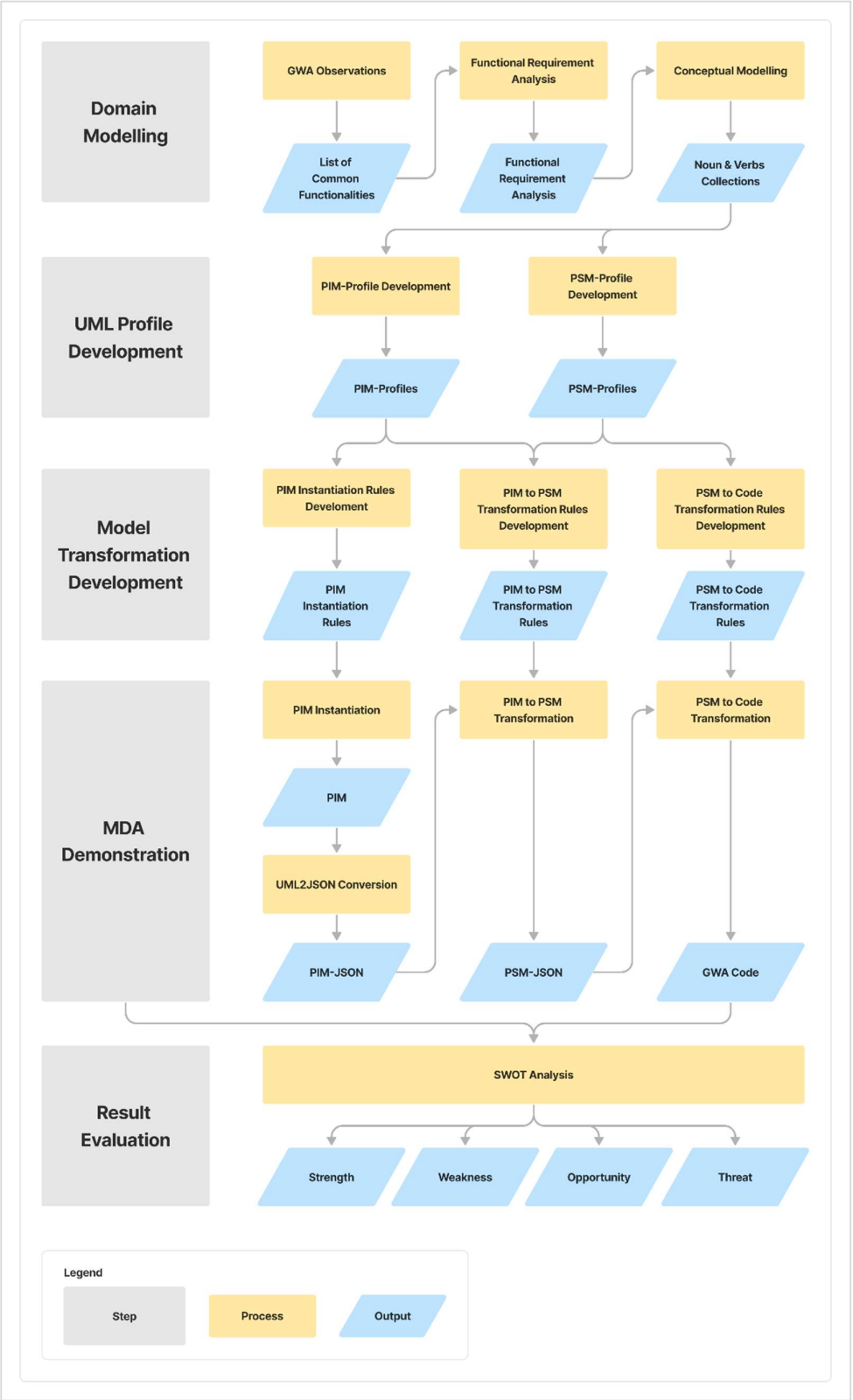


Figure 5. 1 Research Flowchart

User Type #2: The Developer

This user type includes those who are familiar with web application development. For this user type, the approach provides value in the documented steps on how to build a REST-based GWA using the proposed development approach, as well as on how to extend the UML profiles and the proposed MDA-based development methodology. The expectation is that this user type finds the value of this research and improves the development of the proposed methodology which will open to more GWA possibilities that can be developed with this method.

User Type #3: The End-user

This user type includes users who are the target users of the developed GWA. For this user type, the approach provides value in the way that GWA can be produced by the other user types using the proposed methodology. The expectation is that this user type uses the developed GWAs but participates in specifying the requirements.

Figure 5.2 shows the research steps along with the identified user types that might find that particular step valuable. During the start of each step of the research, the expected user type was mentioned to indicate which user type the section was intended to.

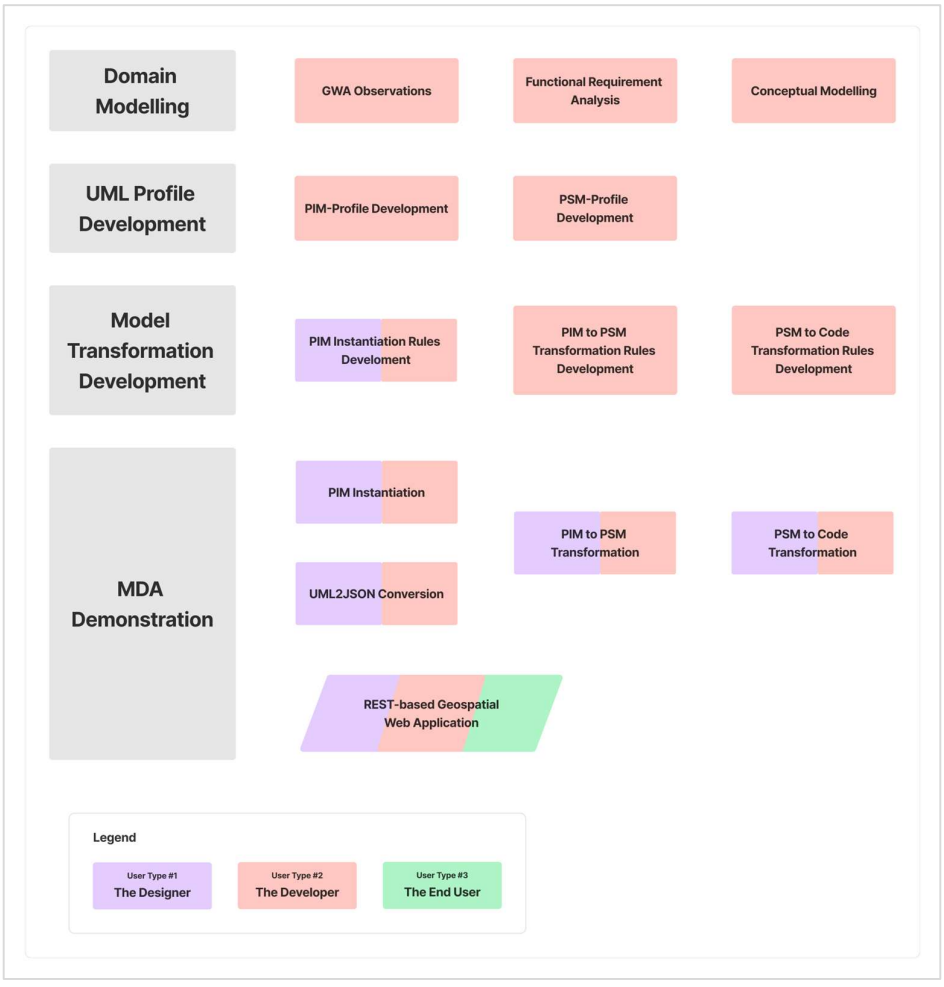


Figure 5. 2 Research process according to target user types

3.1. Domain Modelling

Domain modelling is a process of capturing common requirements of a domain (Wang, 2016). The aim is to understand concepts in a domain and to create representations that explains those concepts. Domain modelling involves defining the scope of the subject, identifying the key concepts and relationships, and creating the representative model to get the full understanding in navigating through the domain. In this research, the scope of the domain modelling step was the domain of GWA.

3.1.1. Geospatial Web Application Observations

The modelling process started by observing multiple interfaces of several modern geospatial web applications. The reason for this decision was to consider the actual user experience in the observation which helps to capture what geospatial web applications have decided to implement from a user perspective. The observation resulted in a collection of functionalities of the chosen GWAs. Functionalities with similar purpose were generalized into one, which were then compiled into a reduced list of common functionalities, which became the input for the next step of the research.

The core considerations on choosing which GWAs to observe were the variety in use cases, data sources and types, license type (open source or proprietary), and the geographic extent (global or local data) of the GWAs. These considerations ensured many varieties of functions are observed in GWAs and made the common function list as representative as possible for GWAs in general.

3.1.2. Functional Requirement Analysis

IEEE Standard 610 (IEEE, 1990) defines requirement analysis as "the process of studying user needs to arrive at a definition of system, hardware, or software requirements." This step elaborates on what are needed to develop a certain part of GWAs. In this research, requirement analysis is applied to the list of common functions from previous step to retrieve the typical functional requirements between client (frontend), server (backend), and data when a user interacts with that function in REST-based GWA. Functional requirement is one of the software requirement types which handle specific functionalities such as calculation, data manipulation, and user interaction in a software or application.

To implement this step, multiple common functions were selected from the common function list. This step considered that the selected common functions to have different types of client-server processes. The selection process is purposed to capture as many varieties of functional requirements as possible.

The selected common functions were described in natural language for their functional requirements. The analyses were taken on the perspective of REST architectural style and OGC API standards where they influenced the description on what client-server components are involved behind the common functions. The result of this step was natural language-based descriptions of functional requirements of the selected common functions which will be used in the next step of UML profile development.

3.1.3. Conceptual Modelling

This step brought the functional requirement descriptions from requirement analysis step as basis to create conceptual model. Noun/verb analysis is performed to capture the nouns and verbs from the description. Noun and verb analyses are two concept mapping methods that are commonly used in natural language analysis (Klavans & Kan, 1998). The purpose of this step is to capture the potential UML class, attribute, operation, and association names to be used in UML profile development step.

The functional requirements were summarized into collections of occurring nouns and verbs, which were then generalized into UML classes using the careful. The hypothesis was the collection of nouns can be potential class and attribute name, while the collection of verbs complements the nouns as potential

operations and association name. The captured list is then observed for the conceptualized UML class representation. The expected result is a collection of potential UML classes that become the basis for creating the real UML profiles used for building PIM and PSM instances.

3.2. UML Profile Development

UML profile development is done by extending existing UML concepts to suit a particular domain (Gómez et al., 2019). In this step, the concept of UML stereotype is used for this extension approach to constructs UML profiles. The expected results of this step are UML profiles that will be used to develop instances of PIM and PSM in MDA implementation.

3.2.1. PIM-Profiles Development

The first step was to develop UML profile that will be used to build PIM, namely "PIM-Profiles" for brevity. The conceptual UML classes from previous sub-step are split based on the technological dependencies, where every UML classes that is interpreted to be independent is used for developing the PIM-Profiles. The approach will be to convert the UML classes from the conceptual model into stereotypes with careful choice of semantics for each class and their related elements.

Model-View-Controller (MVC) design pattern is applied in developing the PIM-Profiles. MVC is a widely used design patten that adopts the principle of separation of concerns, splitting the data representation (Model), interface (View), and interaction functions (Controller) (Syromiatnikov & Weyns, 2014). This separation principle enables simplified complexity and improved reusability of its parts in a development (Lu et al., 2021).

In this context, MVC pattern helps in splitting the research to categorize the stereotypes in consideration to making the model intuitive and bringing convenience in later transformation steps. This approach separates the stereotypes into three (3) UML profiles that are represented in one UML package diagram, a grouping diagram purposed to encapsulate classes, diagrams, or other packages into a higher-level model. The expected result of this sub-step is a package diagram containing PIM-Profiles with MVC categorization.

3.2.2. PSM-Profile Development

Similar approaches in developing the PIM-Profiles model are used to build PSM-Profile. The conceptual UML classes that are interpreted to have technological dependencies become the basis of this sub-step. Afterwards, for each MVC aspect, a package diagram of UML profiles is created. In this sub-step, package diagrams are created to categorize several technology-relates UML profiles (e.g. profile for PostgreSQL, profile for Leaflet) based on MVC design pattern, where they will be represented in another higher-level package diagram.

The expected result of this sub-step is a package diagram containing several package diagrams where technology-related UML profiles are categorized based on MVC design pattern and contains stereotypes related to that technology.

3.3. Model Transformation Development

Model transformation in MDA is the action of transforming a model into another by combining PIM with specific information to result in PSM. In this step, transformation rules were developed and documented to provide a guidance on the proposed model transformation. Examples were also given to each transformation rules. Several categories were determined for the transformation rules: PIM instantiation

phase, PIM-to-PSM transformation phase, and PSM-to-Code transformation phase. The expected results of this step are the documentation of all transformations.

3.3.1. PIM Instantiation

In this sub-step, the rules for develop PIM from the PIM-Profiles' stereotypes were developed and documented. The documentation is expected to guide users in replicating the approach in creating their own instance of PIM. The expected result of this sub-step includes the implementation of developing a PIM instance using PIM-Profiles stereotypes, the PIM instance itself, and the documentation of the workflow.

3.3.2. PIM to PSM Transformation Rules

In this sub-step, the developed PIM instance is taken into transformation step to instances of Platform Specific Model (PSM) which are built upon the PSM-Profiles. The expected result of this sub-step includes the implementation of transforming a PIM instance into PSM, the model transformation process including transformation rules and language used, the PSM instance itself, and the documentation of the workflow.

Transformation rules are developed and introduced to the process, giving constrains and directions as to what should happen in the transformations. Each transformation rule is also demonstrated to transform the PIM instantiation. One important transformation rule assigned in this research is the use of JavaScript Object Notation (JSON) for the diagram. JSON is a light file format for data exchange that is mainly familiarized in web environment. JSON is structured akin to JavaScript object and has its syntax derived from JavaScript, but it is a text-only format where any programming languages can handle this file format.

In this sub-step, one of the transformation rules in this research includes converting the PIM instance into JSON format (called PIM-JSON for brevity) using standardized encoding rules. The purpose of this conversion is to allow automatic model transformation of PIM instance diagrams into PSM diagrams independent of what means the diagrams are being represented. The conversion follows "Best Practice for OGC – UML to JSON Encoding Rules" which is an OGC-initiated approach to allow UML conversion to JSON format for supporting standards managed by OGC. The benefit of this formulated approach is to standardize the way the abstraction models are being represented in JSON format and to ensure transformation rules and languages developed in this research are consistent and reproducible to other users.

Complementary to the JSON transformation rules, the model transformation in this research will be based on JavaScript language in node.js environment, which is not a standardized option. Traditionally, there are various model transformation languages that have been standardized such as Query/View/Transformation (QVT) and ATLAS Transformation Language. However, there is an ongoing debate on the encouragement to use general-purpose programming languages for model transformation such as Java since there has not been a widely acknowledged evidence that indicates the dedicated, traditional model transformation languages perform better (Burgueño et al., 2019). The use of JavaScript for model transformation language in this research entertains the argument of using general purpose programming language for model transformation with several reasons.

JavaScript is known to be tightly related to JSON, which makes it a good choice for the JSON-based model transformation in this research. JavaScript is a prototype-based procedural language that allows object-oriented programming to be performed, which is the paradigm used in model-driven approach as well as UML used in this research. Lastly, JavaScript is popularly used in web application development in

general, which makes it suitable for complementing the nature of MDA implementation in this research which is to perform geospatial web application development.

3.3.3. PSM to Code Transformation Rules

Similar to PIM to PSM transformation, in this sub-step, the developed PSM instance is subjected to model transformation to generate computer code of the aimed geospatial web application. This step also makes use of JavaScript as the model transformation language. A implementation of PSM class transformation to computer code using the developed transformation script is performed. The expected result of this sub-step includes the implementation of the code generation from the PSM, the model transformation process, the resulted code, and the documentation of the workflow.

3.4. MDA Implementation

In this research, MDA is demonstrated to showcase the proof of concept for the developed UML profiles. A GWA example was introduced as the goal of the implementation. Then, the implementation started by leveraging the developed stereotypes within the PIM-Profiles to build an instance of PIM diagram. The diagram is then brought to transformation steps using defined transformation rules into PSM diagram that are built upon stereotypes from PSM-Profiles. Lastly, the PSM diagram is taken into code generation step where it is transformed into computer code of a geospatial web application.

3.5. Result Evaluation

In this research, result evaluation was performed to the developed approach and the result of the implementation. The evaluation was based on the SWOT analysis, which is a popular framework that summarize the Strengths, Weakness, Opportunity, and Threat to identify range of assessments (Helms & Nixon, 2010).

4. DOMAIN MODELLING

This chapter documents the domain modelling step and discusses the results of the step. This step is not performed with reproducibility in mind, but it is possible to reproduce. The most suitable target user to reproduce this step is the User Type #2: The Developer.

4.1. Geospatial Web Application Observation

Six (6) modern GWAs are observed for their common functions. The following subsections elaborated each GWA observation. The summary of the observations is provided in the next subsection.

4.1.1. Google Map

Google Maps is a global-scaled, proprietary web-based mapping service developed by Google. Google Maps serves many purposes including navigation and route, business search, geocoding, street view, and many more. The service uses several types of geospatial data including raster and vector tiles, and vector for user-generated point of interests.

Google Map is selected for observation to represent versatile geospatial web services with global user-base and provide user contribution functionalities to the platform, which provide relatively high number of interactive features in their interfaces. Figure 4.1 show the captured functions in the interface of Google Map.

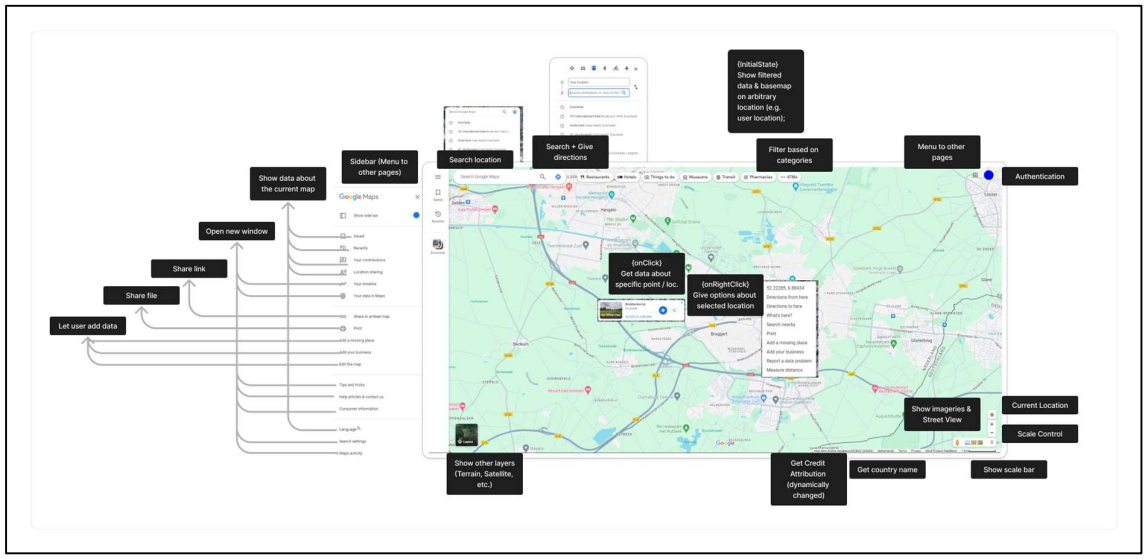


Figure 4. 1 The observed functionalities of Google Map

Based on the observation, there are 16 distinct functions in the Google Map. The platform's most distinct functionalities compared to the other GWAs in this list are the functionality to provide user customized data based on user authentication, and its functionality to provide ranges of data including various types of map tiles, user reviews, geotagged images, and Street View data for any arbitrary location.

4.1.2. OpenStreetMap

Open Street Map is a global-scaled, open-source project for providing free and editable geospatial data of the world. The project provides several possible services including navigation and routing, geocoding, and analysis. The service uses several types of geospatial data such as tiles and vectors.

OpenStreetMap is selected for observation to represent open-source GWAs purposed for huge collaboration initiatives such as citizen science project, which include functionalities with simple interface to easily access data and to promote collaborations. Figure 4.2 shows the captured functions in the interface of OpenStreetMap.

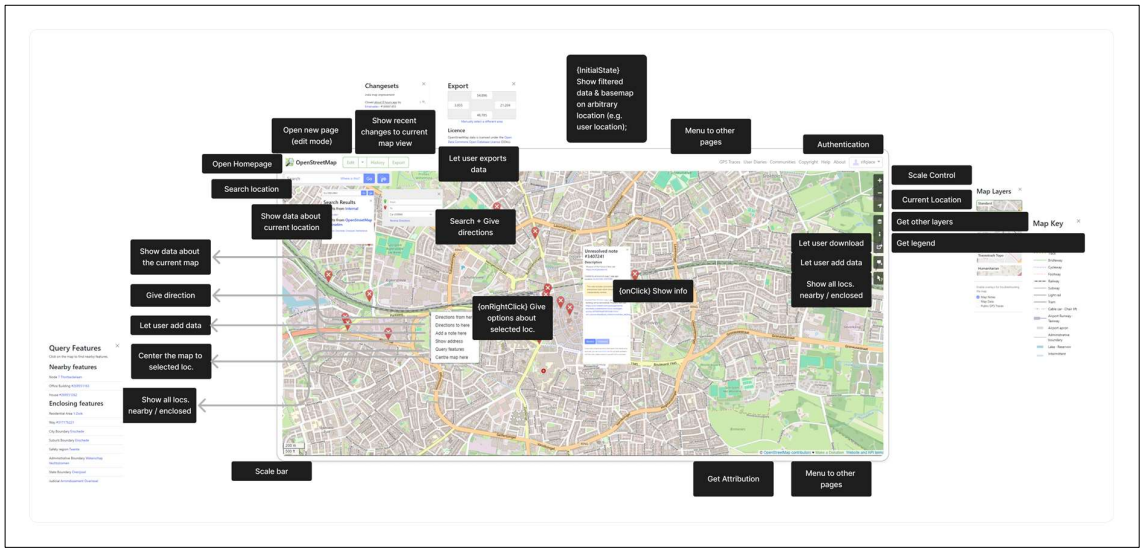


Figure 4. 2 The observed functionalities of OpenStreetMap

Based on the observation, there are 17 distinct functions in the OpenStreetMap. The platform's most distinct functionalities compared to other observed GWAs are to show recent changes on data within the map extent, to perform geospatial process to extract nearby and enclosed data and set of functions to allow user to add new data or edit existing ones.

4.1.3. Earth Explorer

Earth Explorer is a global-based, proprietary geoportal that provides the service to search and download geospatial data and information of the world Earth Resources Observation and Science (EROS) archives by U.S. Geological Survey (USGS). The service provides satellite imagery data to download with various use cases.

Earth Explorer is selected for observation to represent proprietary global-based GWAs which is part of geoportal for providing open access to global data especially satellite imagery data, which have functionalities to perform queries to multiple data sources, to show the queried data, and to provide access to the data. Figure 4.3 shows the captured functions in the interface of Earth Explorer.

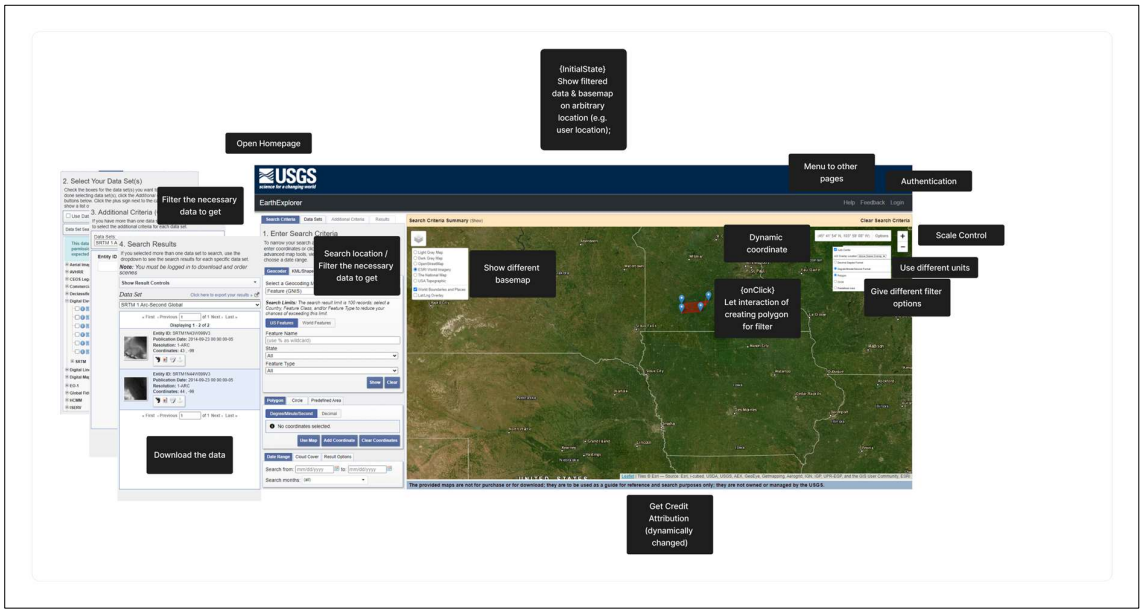


Figure 4. 3 The observed functionalities of Earth Explorer

Based on the observation, there are 13 distinct functions in the Earth Explorer. The platform's most distinct functionalities compared to other observed GWAs are advanced filtering options, displaying preview of selected data coming from other sources and providing access to download data in different file formats.

4.1.4. Samenmeten Data Portal

Samenmeten data portal is a Dutch national-based, open-source citizen science platform maintained by Dutch National Institute for Health and Environment (RIVM) that provide measurements of air quality and noise level for public use. The platform provides visualization of sensor locations and the measurements and API for accessing the measurement data.

Samenmeten data portal is selected for observation to represent a local-based geoportal visualization that display citizen science data of various types (air quality parameters) and provide access to the data via API connection, which allows interactivity to data for showing advanced and various types of information. Figure 4.4 shows the captured functions in the interface of Samenmeten data portal.

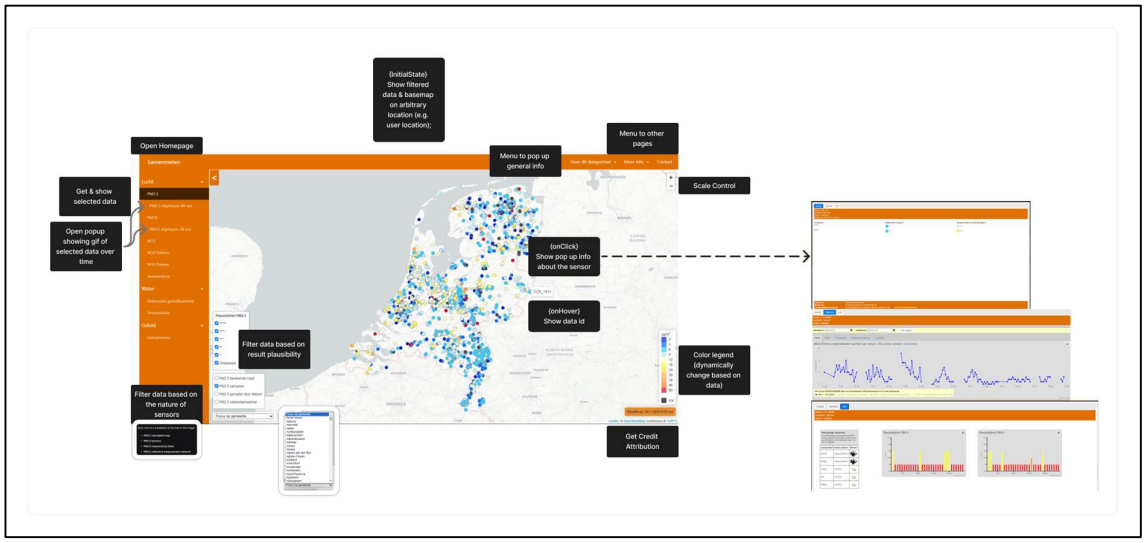


Figure 4. 4 The observed functionalities of Samenmeten

Based on the observation, there are 9 distinct functions in the Samenmeten data portal. The platform's most distinct functionalities compared to other observed GWAs are showing popup that open various capabilities of the selected data, including displaying metadata, processing the data transformation into static and interactive charts, and temporal- and attribute-based filtering function to the selected data.

4.1.5. ShadeMap

ShadeMap is a global-based, proprietary platform that provides simulation of shadows and displays them on the map. It provides free access to machine learning-based shadow simulation for most of the worlds and the more accurate LiDAR-based simulation for its premium users.

ShadeMap is selected for observation to represent proprietary geospatial web service that specialized in displaying 3D data with on-the-fly data transformation process (shadow simulation) that can handle interactive input from user, which contain visualization that showcase the main value of the web service and functionalities that allows user to control the data visualization which can be limited based on the tier. Figure 4.5 shows the captured functions in the interface of ShadeMap.

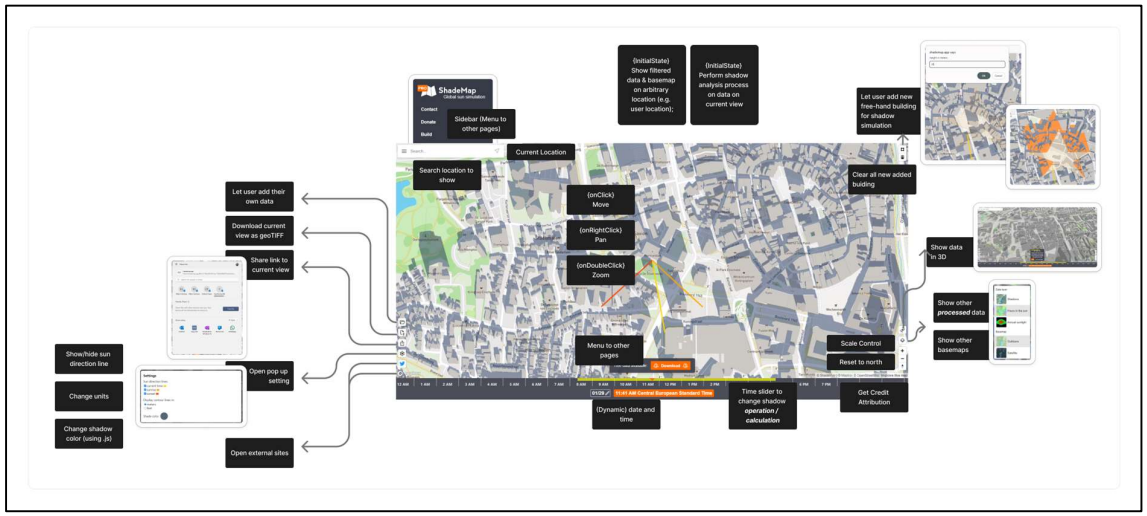


Figure 4. 5 The observed functionalities of ShadeMap

Based on the observation, there are 15 distinct functions in the ShadeMap. The platform's most distinct functionalities compared to other observed GWAs are providing 3D data visualization, providing on-the-fly geospatial process to existing data and user-provided data that is not retained, changing the styles of the shadow, and allowing bounding box filter of displayed data to select which area to perform the geospatial process.

4.1.6. PDOK Viewer

PDOK Viewer is a Dutch national-based, proprietary platform that displays the open government geo-information datasets of The Netherlands. PDOK geoportal provides access to varieties of data via API using various services such as WMS, WMTS, and WFS.

PDOK Viewer is selected for observation to represent proprietary local-based GWAs that is part of geoportal that showcase various kind of data for public access, which contain simple interface for data visualization that connects to multiple data source with limited interactivity. Figure 4.6 shows the captured functions in the interface of PDOK Viewer.

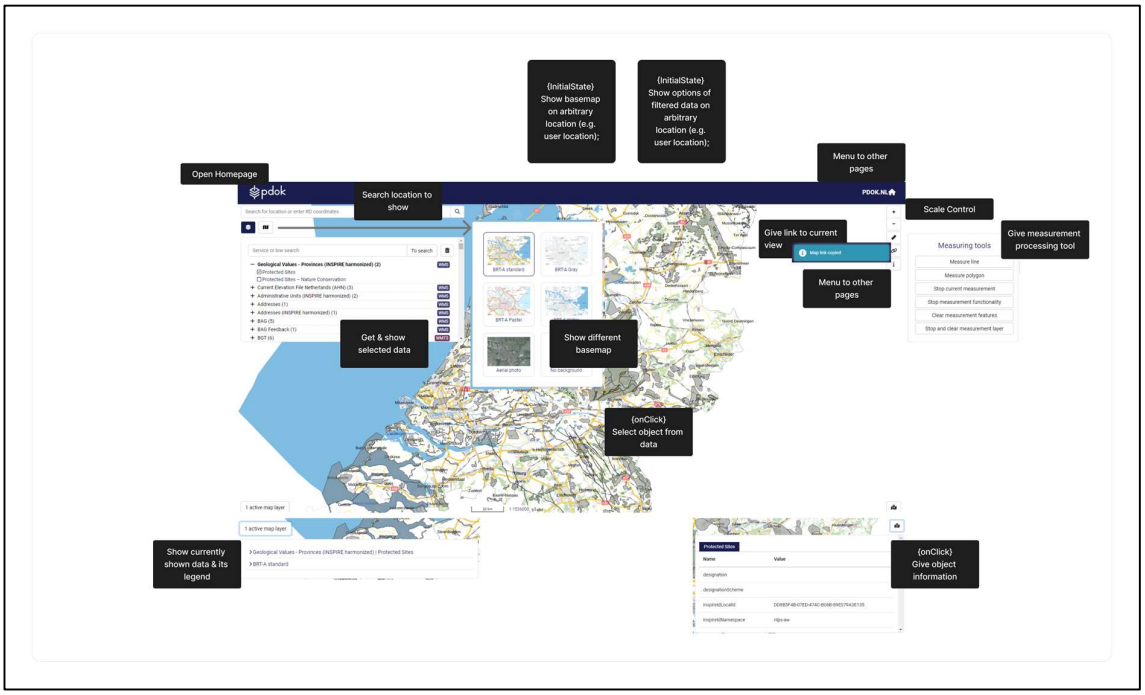


Figure 4. 6 The observed functionalities of PDOK Viewer

Based on the observation, there are 10 distinct functions in the PDOK Viewer. The platform's most distinct functionalities compared to other observed GWAs are functionalities to provide big number of data sources to retrieve and display data and show information based on selected data which might come from different data sources.

4.2. Common Functionalities Identification

Based on the observations of the GWAs, the captured functions are generalized into several distinct categories of functions. The observations captured 21 distinct categories of functions. Table 4.1 shows the observed functions and the categories each falls into.

Search for Location

This function closely relates to data search capability of the GWA and display the result on the map.

Perform Processes

This function contains various processes including giving direction between data and points of interest, performing measurement, and more specific geospatial processes such as chart generation and shadow analysis.

Open Other Resources

This function relates to directing user to different pages, which commonly displayed as menu, sidebar, and homepage button.

Authentication

This function relates to user authentication that can have influence in which data are available to an arbitrary user and which services are accessible.

Attribution

This function relates to accessing metadata of displayed data, which commonly found in GWA with licensed data that might come from connected external data sources.

Get Other Layers

This function relates to accessing and displaying available data, especially basemap.

Initialization

This function relates to all data displayed and services run during the first time the GWA is accessed, which mainly includes show filtered data and show pre-defined location.

Filter Data to Show

This function relates to data filter and query capabilities to either spatial or non-spatial data in the GWA. This function can form in various approaches such as submitted search and pre-defined buttons.

Show User's Current Location

The function relates to capability to use current user location determined by GPS of the user's device and display the location on the GWA.

Scale Control

This function relates to zooming capabilities in the GWA which can be in form of button or mouse scroll.

Scale Bar

This function relates to visualization of current scale in form of bar, which is also commonly complemented with scale number the bar represents.

Get Info About Selected Location

The function relates to showing info about an arbitrary data or point of interest which mainly in form of popup when user interact with the displayed data or point of interest.

Let User Add Data

This function relates to the capability for user to add new data to the GWA with two approaches on whether the data is retained to the GWA or not.

Share Link

The function relates to the capability to share link of the current view of the map.

Download Data

The function relates to downloading, exporting, and sharing capabilities in the GWA.

Add Legend

The function relates to the legend of the map display in GWA, commonly shown as part of the map or as a separate user interface component, with options to hide it.

Custom Right-Click Interaction

The function relates to the right-click interactivity that shows the GWA-customized context menu when clicked to certain user interface component.

Use Different Units

The function relates to the capability to change the units of the map use such as in the scale bar and distance measurement.

Change Style

The function relates to the capabilities for user to change the style of components in the interface, such as modifying the shadow colour.

Interact With Currently Shown View

The function relates to the capabilities for the user to perform advanced interaction with the GWA's user interface such as creating bounding box or perform multiple selections.

Move or Orient Current View

The function relates to the capabilities to move the current view of the GWA map such as basic map panning and rotating as well as dedicated functionalities like go to pre-defined centre of the map and orient the north direction.

Table 4. 1 The observed common functionalities from the multiple GWAs

Requirements	Google Map	Open Street Map	Earth Explorer	Samen Meten	Shade Map	PDOK
Show location	Search location	Search location	Search location		Search location to show	Search location to show
Perform processes	Search + Give directions Give direction to / from selected location Perform measurement using selected location	Search + Give directions Give direction		Show pop up info about the sensor + analytics	Show other processed data Time slider to change shadow operation / calculation Show/hide sun direction line Perform shadow analysis process on data on current view	Give measurement processing tool
Open Other Resources	Menu to other pages Open Homepage Open new window Sidebar (Menu to other pages)	Menu to other pages Open Homepage	Menu to other pages Open Homepage	Menu to other pages Menu to pop up general info Open popup showing gif of selected data over time	Menu to other pages Open pop up setting Open external sites Sidebar (Menu to other pages)	Menu to other pages
Authentication	Authentication	Authentication	Authentication			
Attribution	Get Credit Attribution (dynamically changed)	Get Attribution	Get Credit Attribution (dynamically changed)	Get Credit Attribution	Get Credit Attribution	
Get Other Layer	Show other layers	Get other layers	Show different basemap	Show other basemaps	Show other basemaps Show data in 3D	Show different basemap
Initialization	Show filtered data & basemap on arbitrary location	Show filtered data & basemap on arbitrary location	Show filtered data & basemap on arbitrary location	Show filtered data & basemap on arbitrary location	Show filtered data & basemap on arbitrary location	Show options of filtered data on arbitrary location (e.g. user location) Show basemap on arbitrary location (e.g. user location)
Filter Data to Show	Filter based on categories Show filtered data on / around selected location	Show data about the current map	Filter the necessary data to get based on area / name filter Give different filter options	Filter data based on result plausibility Filter data based on the nature of sensors		
Show user's current location	Current Location	Current Location			Current Location	
Scale control	Scale Control	Scale Control	Scale Control	Scale Control	Scale Control	Scale Control
Scale bar	Show scale bar	Scale bar				
Get info about selected location	Get data about specific point / location Show data about the current map Show image data & Street View Get country name	Show recent changes to current map view Show all locs. nearby / enclosed Show info about current location Show info about selected data	Dynamic coordinate	Get & show selected data Show data id	(Dynamic) date and time	Get & show selected data Select object from dataset Give object information
Let User Add Data	Let user add data Add new data on selected location	Let user add data Open new page (edit mode) Let user add data			Let user add their own data	
Share Link	Share link				Share link to current view	Give link to current view
Download Data	Share file	Let user exports data Let user download	Download the data		Download current view as geoTIFF	
Add Legend		Get legend		Color legend (dynamically change based on data)		Show currently shown data & its legend
Custom right-click	Give options about selected location	Give options about selected loc.				
Use different units			Use different units		Change units	
Change style					Change shadow color	
Interact with currently-shown view			Let interaction of creating polygon for filter		Let user add new free-hand building for shadow simulation Clear all new added buiding	
Move / orient current view		Center the map to selected loc.				Reset to north

4.3. Functional Requirement Analysis

This section documents the requirement analysis step and discusses the results of the step. The selected functions from GWA observation step were analysed and described using natural language for its typical client-server processes. From the list of common functions from the domain modelling step, three functions are selected for this requirement analysis:

1. Search for Location.
2. Perform Process, in this case distance measurement process.
3. Get Other Layers, in this case to get the basemaps.

This selection considers that the chosen functions cover different type of web app logics behind them, which are elaborated in the next subsection. The following subsections showcase the reasoning behind each selected function and the analysis step that were performed.

4.3.1. Search for Location

Search functionality is a functionality in a GWA that is purposed for getting data from the connected data source and use the data in the GWA interface. This function is selected for functional requirement analysis because it is a good function example that can showcase the typical frontend-backend-data source connection in a GWA.

The implementations usually involve a search bar provided on the frontend interface of the GWA where users input the location-based query into the search bar and submit it, which can be the name of the location or spatial search such as a pair of coordinates or bounding box extent. The input is sent to the backend of GWA which is then handled to use the query into the data source before the queried data are being sent back to the frontend side to be used.

In the case of REST-based GWA with compliance to OGC API standards, several points can be specified. One of the important practical points for both compliances is the use of HTTP as the protocol to handle the communication in web environment. Any communication between server (backend) and backend of the GWA is highly suggested to use HTTP messages containing the HTTP method and status, HTTP header, and message body (MDN, n.d.). In this case, search queries and their results are being transformed into HTTP message in any client-server communications.

Another point to consider is the use of JSON object. JSON becomes the most preferable format for data exchange in REST implementation (Soni & Ranga, 2019). OGC API standards with its overlying OpenAPI specification also heavily implement JSON schema for their APIs. In this case, HTTP messages between client and server is suggested to use JSON format.

Below are the summarized functional requirements for the search location functionality:

1. The user enters a location-based query into the search bar. It could take various forms:
    - Keyword search: the name of a place e.g. city.
    - Spatial search: a specific coordinate or a bounding box.
2. The user initiates the search by pressing "Enter" or clicking a search button.
3. The front-end code constructs an HTTP GET request with appropriate query parameters.
4. The backend server parses the HTTP request to extract relevant information from the query parameters.
5. The request is routed to the specific geospatial module or service within the backend, such as a geospatial search service or a geocoding service.
6. The geospatial module or service executes the necessary operations based on the type of geospatial search requested.

7. Geospatial services may interact with the geospatial data source to retrieve relevant information.
8. Once the data is retrieved, the backend performs additional processing, such as transforming the raw data into GeoJSON format.
9. The geospatial service generates the HTTP response in a standardized format (e.g. JSON or GeoJSON) and sends it back to the front end using compatible parsing libraries.
10. The front end parses the HTTP response to extract the relevant information.
11. The front end processes the retrieved information to update the user interface with the search results.

4.3.2. Distance Measurement

Distance measurement is a feature in GWAs that is purposed for measuring the distance between data or point of interests. This function is selected for functional requirements analysis because it is one of the most common geospatial processes implemented in GWA, which can help in showcasing how a process typically work in a GWA.

The implementation usually starts with an initiation by the user to start a measurement, commonly by clicking a dedicated button or performing a right-click to a data or point of interest where an option to measure distance is available in the context menu. The measurement type can also vary depending on whether it is a straight-line measurement or one that include routing service. After the user select the data, HTTP message is sent to server containing all information required for performing all required processes. This initiates a creation of 'job' which is a resource that represent any kind of processes. This job resource contains metadata of the process including ID, involved process, and status. The processes are executed where they also handle the status of the relevant job. Until the status of the job is declared as finished, the results of the processes are being sent to the client using HTTP message where it will be used at the frontend of the GWA.

In the case of REST-based GWA with compliance to OGC API standards, the uses of HTTP and JSON are still applied. A specific point to be added to both compliances is the use of 'job' and all other concepts around it which comes from the OGC API – Processes.

Below are the summarized functional requirements for the distance measurement functionality:

1. The user initiates a distance measurement process by interacting with provided interface.
2. The user selects two points on the map for distance measurement. Event listener attached to the interface detects this interaction.
3. The frontend sends a POST request to the backend server to initiate the distance measurement. The request contains a JSON payload with the selected points and measurement parameters.
4. The backend server parses the HTTP request to extract relevant information. The server validates the input data.
5. The backend creates a job resource to handle the distance measurement task. A unique identifier is assigned to the job. The server responds with a 202 Accepted status and a URL to poll for job status.
6. The distance measurement is performed by the appropriate module or service.
7. The frontend periodically checks the status of the job by sending a GET request to the job URL.
8. When the distance measurement is complete, the backend updates the job resource with the result.
9. The frontend updates the user interface to display the measured distance.

4.3.3. Select Basemap Layer

Basemap layer option is a common feature in GWAs that is purposed for providing different types of basemap layer for GWA map display. This function is selected for functional requirements analysis because it is a simple functionality in GWA that can showcase how an interaction in a component can affect another component.

The implementation usually starts from initialization where a dedicated user interface component for showing the basemap options are presented. When the GWA is initialized, a HTTP exchange is happened to extract the available basemap options from the data source. This communication commonly involves external APIs to get the basemap layer as well as the metadata such as the preview, name, and provider attribution. When a user selects an option of the provided basemap layers, a HTTP message is sent from the client to server which retrieves the data from the data source. This basemap layer data can be in many file formats including raster image, map tiles and vector tiles. After the data is retrieved and is sent back to the client frontend, the data is rendered and update the previous map display.

In the case of REST-based GWA with compliance to OGC API standards, the uses of HTTP and JSON are still applied. Depending on the file format that is used, several OGC API standards can apply. However, the concept of 'collection' which is implemented by OGC API – Core is applied to represent any kind of data. The concept of 'collection' that treat everything as a resource regardless of the file format helps in enforcing REST architectural pattern to the GWA while also maintain the interoperability to the compliant GWA.

Below are the summarized functional requirements for the basemap layer selection functionality:

1. During the initialization of map visualization in the application, a section of the UI provides an option list or dropdown menu for selecting basemap layers. The frontend sends an HTTP GET request to the backend for metadata of the available basemap layers and displays the response in a dedicated section.
2. When a user selects a basemap layer from the dropdown, the frontend sends an HTTP GET request to the backend, specifying the selected basemap layer.
3. The backend parses the HTTP request and fetches the data associated with the selected basemap layer.
4. The backend sends the requested data (or alternatively tiles) back to the frontend by generating HTTP 200 response.
5. The frontend parses the HTTP response to retrieve the data or tiles from the backend.
6. The frontend updates the map, replacing the current basemap layer with the newly selected one. If the backend provided raw data, the frontend might need to render the basemap layer.
7. The user can interact with the map, such as panning or zooming. The frontend sends additional GET requests to the backend for tiles covering the newly visible area.

4.4. Conceptual Modelling

This step conceptualizes the functional requirements using the concept of UML class. Since REST principle and OGC API standards have been applied as the perspective in analysing the functional requirements, the UML classes resulted from this step should also capture the requirements that allow development of GWAs that abide with the REST principle and OGC API standards.

First, the noun/verb analysis was performed. Table 4.2 shows the summary of all nouns and verbs. Based on the analysis and considerations to each item in the summary table, several UML classes are generated as shown in Figure 4.7. The classes are intentionally structured in three separate columns to apply MVC

design pattern for separation of concerns. The UML classes resulted in this phase are not using many UML capabilities such as multiplicity and associations, which will be applied during the development of UML profile.

Table 4. 2 The list of collected nouns and verbs for potential UML class semantics.

Noun	Verb
User Interaction	enter (Location-based query) initiates (Search process; Distance measurement process) press (Enter key) click (Seach button) interact (Provided Interface) select (Displayed Data on Map; provided Basemap layer)
Front End	construct (HTTP GET request) parse (HTTP response) extract (Data; from HTTP response) process (Data) update (Interface) attach (Event listener) send (HTTP request) initiate (Search process; Distance measurement process) check (Job's status) display (Job's result; Metadata of basemap layer options) specify (Selected Basemap layer)
Interface Type: Search bar, Button, Map, Section	provide (Basemap layers option list)
Event listener	detect (Interaction)
Location-based query Type: Keyword search, Spatial search Query parameter	
Search Process	
Back End Server	parse (HTTP request) extract (Data; from Query parameter in HTTP request) route (HTTP request) validate (extracted Data; from HTTP request) create (Job) respond (to Job) update (Job's result)
Geospatial Module / Service	execute (Geospatial operation) interact (with Data Source) retrieve (Data) generate (HTTP response) send (HTTP response; to Front end)
Job Unique identifier, Status, URL, Result	handle (Distance measurement process)
Geospatial Operation Example: Distance measurement process	
HTTP Request Type: GET, POST Format: JSON, GeoJSON Content: Data (e.g. Point, Measurement parameter)	
HTTP Response Format: JSON, GeoJSON	
Data Source	
Data / Information	
Basemap Layer	
Metadata Example: Basemap layer options	

Figure 4. 7 The conceptualized class from the described functional requirements.

Below are the descriptions of each conceptualized class:

1. The *UserInteraction* class represents interactions that are captured from the functional requirements. It covers all possible interactions a user might perform with a geospatial web application. The *interactionType* attribute provide the mean to specify what kind of interaction a user perform in any arbitrary instance of this class, examples from the functional requirements are *click* to represent any kind of mouse clicks and *keypress* to represent any kind of keyboard presses. The *enter* operation represents the possibility of a user to enter an input to geospatial web application. The *interact* operation represents the possibility of a user to interact with UI elements of geospatial web application and perform any interaction types that are defined in the *interactionType* attribute. The *initiate* operation represents the possibility of a user to perform a process by interacting with provided UI component in the geospatial web application interface.
2. The *Frontend* class represents the various frontend components of the geospatial web application that are captured from the functional requirements. It covers the different types of interfaces available to users. The *interfaceType* attribute specifies the observed types of interfaces from the requirement analysis, which includes options such as searcher, button, map, and container. The operations defined within this class covers interactions and data handling on the client side. The *attach* operation allows the attachment of event listeners to view components, enabling user interactivity with the frontend component. The *display* operation represents the rendering of data on the frontend interface to visually show the information. The *construct* and *parse* operations represent the HTTP communication management for building request objects and interpreting response objects. The *extract* operation represents the action to pulling specific data from HTTP messages. The *specify* operation represents the action of defining which data will be used or displayed while the *process* operation handles the manipulation and preparation of this data. Lastly,

the *update* operation represents the action of the frontend components updating themselves with new data to provide dynamic and interactive user experience.

3. The *HttpProtocol* class represents the mechanisms and operations associated with handling HTTP communications within the geospatial web application. It is responsible for managing the flow of HTTP requests and responses between the frontend and backend component. The *messageType* attribute specifies whether a particular instance of an HTTP message is a request or a response. The *send* operation represents for the action of transmitting an HTTP message, such as a request from the frontend to the backend or a response from the backend to the frontend. The *receive* operation represents the reception of those HTTP messages.
4. The *BackendServer* class represents the server-side component which handles operations within a GWA. The *parse* operation represents the interpretation of incoming HTTP requests by the server, while the *extract* operation represents the retrieval of data from these requests. The *route* operation represents the routing action where requests are sent to the appropriate handler within the server. The *construct* operation represents the action of forming the HTTP response object to be sent back to the client. The *create* operation represents the action of creating a job resource for an initiated process within the server. The *update* operation represents the action of updating the existing job resource based on the ongoing process that takes place, such as adding the job resource with the process result and updating the status of the job. The *validate* operation represents the action of server ensuring that all exchanged data complies to the predefined standards.
5. The *Job* class represents a discrete resource of an initiated process within the backend server of a GWA. This class represents the mechanism responsible for managing and tracking individual processes that need to be executed. The *uniqueIdentifier* attribute represents the distinct ID for each job and the status operation represents the current state of the job. The *handle* operation represents the execution of the job by handling the specified process and producing the necessary output.
6. The *BackendService* class represents the part of backend server that executes processes and interacts with data sources. The *input* and *output* attributes represent the data received and produced by the service. The *execute* operation represent the action of performing the process associated with the service. The *interact* operation handles interaction with the data source. The *retrieve* operation represents the action of obtaining data from the data source to be processed.
7. The *DataSource* class represents the location of where the data required by the GWA are stored. Based on the observation and analysis in this research, there is no attribute and operation that is relevant to this class.
8. The *Data* class represents the actual data used or managed by the GWA. Based on the observation and analysis in this research, there is no attribute and operation that is relevant to this class.

The conceptual classes above can be categorized into two categories, based on the interpreted level of abstractions for this research. This categorization is also an application of separation of concern concept, where each class is interpreted on whether it is more valuable to be used in building PIM-Profiles or PSM-Profile.

The classes used in the PIM-Profiles model are as follows:

1. UserInteraction: This class represents a high-level, fundamental part of the application's requirements which is separated from the concern of what technology that is being interacted with.

2. Frontend: This class focuses on the types of elements and their relevant interactions, which are concepts that remain consistent regardless of the chosen technology.
3. DataSource: This class represents the abstraction of data access without specifying the underlying data store technology.
4. Data: This class represents a high-level abstraction of what data is needed and how it is structured, which is independent of how the data is stored and processed on a specific platform.

The classes used in the PSM-Profile are as follows:

1. BackendServer: This class deals with the handling of HTTP exchanges that bridges the data source and frontend of the GWA, which is not something to be concerned about in PIM-Profiles development as it depends on the specific technology being chosen in the GWA development.
2. BackendService: This class is closely related to the BackendServer class which is not within the scope of PIM-Profiles building and is dependent on the chosen technology.
3. Job: This class is managed in the server side of the GWA which makes it closely related to BackendServer.
4. HttpProtocol: This class represents a specific technology for the GWA which makes it a technology-specific concept.

The summarized and interpreted conceptual classes above become the result that is used in the next step of building the UML profiles. The existence of the classes, attributes, and operations, as well as the absence of them, are considered in the next step where the classes are engineered to be suitable for practical implementation in the MDA implementation.

5. UML PROFILE AND MODEL TRANSFORMATION DEVELOPMENT

This chapter explains the development of UML profiles and the model transformation rules for the MDA implementation. There were two UML profiles to represent the MDA level of abstractions: Platform Independent Model (PIM) and Platform Specific Model (PSM). Afterwards, the elements of PIM-Profiles were mapped into the PSM-Profile of the selected technologies, which resulted in the PIM-to-PSM model transformation. Lastly, the PSM-Profiles were mapped into the computer code of the selected platforms to develop the PSM-to-code transformation rules.

All steps in this chapter were taken and documented with their reproducibility in mind. The appropriate user types were mentioned at the start of each section.

5.1. PIM Profile Development

This step designs the UML conceptual classes into both PIM-Profiles. The development started with analysing each of the conceptual classes using the perspective of MVC design pattern. Each class from the conceptual model was categorized based on whether it is closely related to the data layer (Model), the interface layer (View), or the intermediary layer (Controller).

This step was performed with reproducibility in mind. The most suitable target users to reproduce this step are the User Type #2: the Developers. By knowing how to build the PIM-Profiles, users can reproduce the step to build their own UML profile for building the PIM or to improve the existing ones.

After the categorization, each class was expanded into more specific concepts that are relevant to the GWA development using the UML stereotype. This approach was performed by specifying the right semantics for each stereotype and determining the attributes and operations within the stereotype, while also maintaining the whole profile diagram in the same abstraction level.

Figure 5.3 displays the UML package diagram showcasing the implementation of MVC separation of concerns in the PIM profile development. The top rectangle *Pim* represents a package for the PIM itself while the apply arrows represent the direct relationships that showcase which profiles are being applied to the package. The three profile rectangles represent the MVC-separated PIM-Profiles:

1. PimModelCore profile for data related stereotypes (Model).
2. PimViewCore profile for interface related stereotypes (View).
3. PimControllerCore profile for data-interface mediatory stereotypes (Controller).

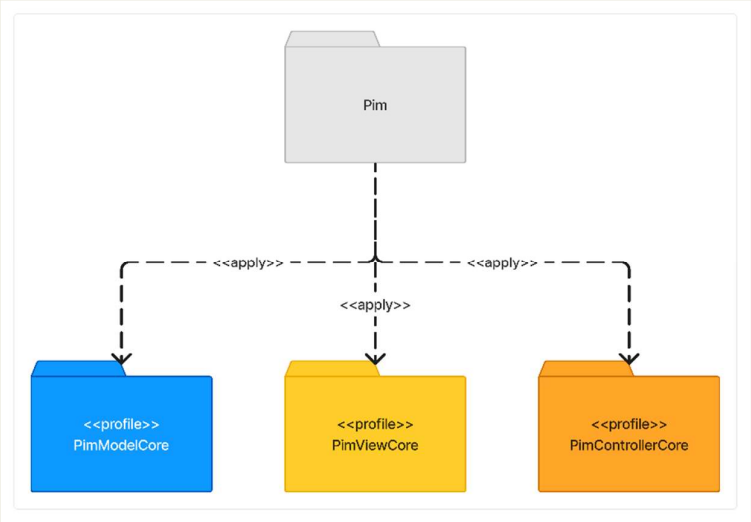


Figure 5. 3 The structure of the proposed PIM-Profiles based on MVC design pattern.

Additional point to be mentioned is the use of colours in the PIM classes as well as the PSM classes shown after the transformation during the MDA implementation. The purpose of the colours is to indicate which MVC pattern the PIM class is related to like shown in the Figure 5.3 above. There are four types of colours being used.

In the UML profile development, two UML metaclasses were extended into stereotypes: Class and Association. During the practical implementation of UML profiles, the Class stereotype was used to contain information related to relevant classes within the profile, while the Association stereotypes are responsible to relate a Class stereotype with another Class stereotype from both the same and a different profile. The following subsections elaborate on each profile.

5.1.1. PIM MVC-Model Profile

Figure 5.4 shows the contents of PimModelCore profile which handles data-related concepts. Both Class and Association metaclasses are pointed with arrow that came from the stereotypes, indicating an extension relationship. Each stereotype may also be related to each other which is shown with bold arrow that indicates an inheritance relationship. The following paragraphs describe each stereotype in detail.

Dataset Class Stereotype

- This class stereotype represents the collections of data used in GWA that are organized in a specific format namely as vector or raster data format.
- This class stereotype does not inherit, nor does it have public attributes or operations to be inherited by other class stereotypes.
- This stereotype has an inheritance relationship with two class stereosubtypes: *SpatialDataset* for spatial data and *NonSpatialDataset* for non-spatial data. These two stereotypes are addressed separately because data with spatial properties may have different data structure compared to non-spatial data and may be treated differently.

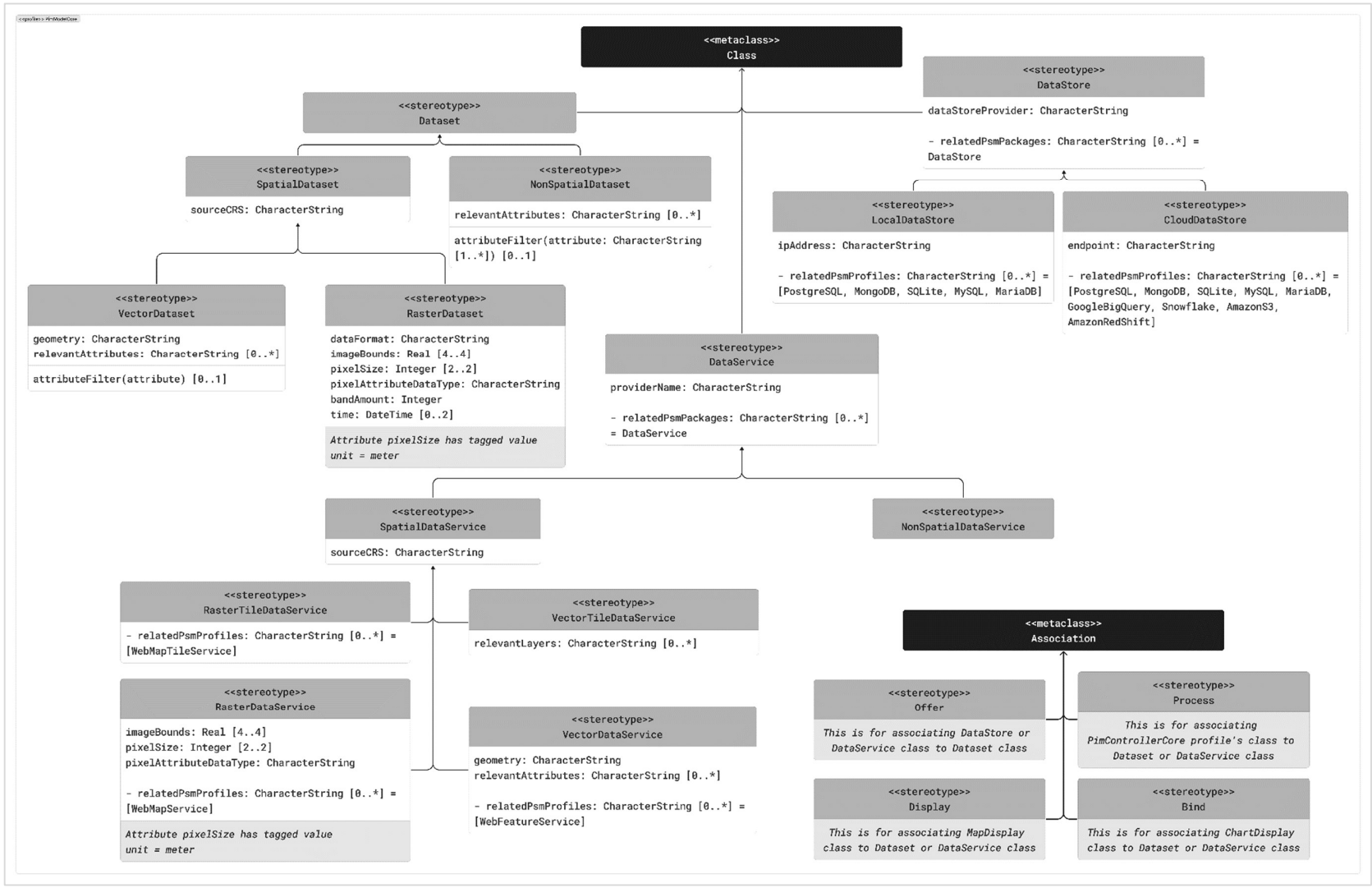


Figure 5. 4 The PimModelCore profile stereotypes

SpatialDataset Class Stereotype

- This class stereotype represents any collection of data that contains a spatial property. This includes spatiotemporal vector and raster data.
- *SourceCRS* attribute: a string that provides the crucial metadata regarding the original coordinate reference system of the data, which helps to know how the data can be accurately located on the surface of the Earth, as well as to know whether a coordinate transformation is needed for the GWA implementation.
- This class stereotype has inheritance relationships with two more specific stereotypes: *VectorDataset* for vector data and *RasterDataset* for raster data. These two specific stereotypes are addressed separately because they have different data structure and different uses in how they represent the real world.

NonSpatialDataset Class Stereotype

- This class stereotype represents any collection of data that contains no spatial property.
- *relevantAttributes* attribute: an array of strings referring to data properties that have meaningful role in the GWA based on user preference.
- *attributeFilter* operation: a function to filter data based on selected attributes.
- This class stereotype inherits attributes and operations from the *Dataset* class stereotype.
- This class stereotype has inheritance relationships with more specific stereosubtypes: *VectorDataset* for vector data and *RasterDataset* for raster data. These two specific stereotypes are addressed separately because they have different data structure and different use cases in how they represent the real world.

VectorDataset Class Stereotype

- This class stereotype represents data with the vector data model, which stores discrete features with defined shapes and boundaries.
- *geometry* attribute: a string that specified geometry type defined by ISO 19107:2003.
- *relevantAttributes* attribute: an array of strings referring to data properties that have meaningful role in the GWA based on the user preference.
- *attributeFilter* operation: a function to filter data based on selected attributes.
- This class stereotype inherits attributes and operations from the *SpatialDataset* and *Dataset* class stereotypes.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

RasterDataset Class Stereotype

- This class stereotype represents data with the raster data model, which stores data as a grid of regularly sized pixels.
- *dataFormat* attribute: a string of the specified format in which the raster data is stored.
- *imageBounds* attribute: an array of coordinates providing to the geographic extent of the raster data.
- *pixelSize* attribute: an array of integer values that represent the size of a single pixel in the raster data. This attribute has a tagged value of *unit* to indicate the unit of the size.
- *pixelAttributeDataType* attribute: a string that define the data type for the pixel value.
- This class stereotype inherits attributes and operations from the *SpatialDataset* and *Dataset* class stereotypes.

- *bandAmount* attribute: an integer that refer to the number of bands within the raster data.
- *time* attribute: an array of data time that refer to the temporal extent of the raster dataset.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

Below is the elaboration for Dataset class stereotypes and its stereosubtypes. DataStore is different from the Dataset because DataStore refers to the system for storing data, while Dataset refers to the collection of data that may be stored in data stores.

DataStore Class Stereotype

- This class stereotype represents a generic mechanism for storing and retrieving data.
- *dataStoreProvider* attribute: a string of the specific data store provider that is being used by the user.
- This class stereotype does not inherit attributes or operations from other class stereotypes.
- This class stereotype has inheritance relationships with two specific stereosubtypes: *LocalDataStore* for data stores on the local machine and *CloudDataStore* for data stores in a remote cloud environment. These two stereotypes are defined separately because they have different implementations for data access and data persistence.

LocalDataStore Class Stereotype

- This class stereotype represents data stores that reside on the local machine.
- *ipAddress* attribute: a string that provides the IP address of the data store. This is required simply to emphasize that even within local data store, an IP address is an important information.
- This class stereotype inherits attributes and operations from the *DataStore* class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

CloudDataStore Class Stereotype

- This class stereotype represents a data store that is located in some remote cloud environment.
- *endpoint* attribute: a string that provides the URL or identifier of the data store service.
- This class stereotype inherits attributes and operations from the *DataStore* class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

Below is the elaboration for DataService class stereotypes and its stereosubtypes. DataService is different from the DataStore because DataStore refers to the system for storing data, while DataService refers to a service that provide access to data which may be stored in a data store.

DataService Class Stereotype

- This class stereotype represents a service that provides access to data.
- *providerName* attribute: a string that specifies the name or identifier of the service that provides the data.
- This class stereotype does not inherit attributes or operations from other class stereotypes.
- This class stereotype has inheritance relationships with two specific stereotypes: *SpatialDataService* for data sources that contain geospatial information and *NonSpatialDataService* for data sources that contain only non-geospatial information. These two specific stereotypes are addressed separately because each type of data service may have different means in accessing their services.

NonSpatialDataService Class Stereotype

- This class stereotype represents data services that provide access to non-spatial data.
- This class stereotype does not inherit, nor does it have public attributes or operations to be inherited by other class stereotypes.

SpatialDataService Class Stereotype

- This class stereotype represents data services that provide access to spatial data.
- *sourceCRS* attribute: a string that provides the crucial metadata regarding the original coordinate reference system of the data provided by the service.
- This class stereotype inherits attributes and operations from the *DataService* class stereotype.
- This class stereotype has inheritance relationships with four specific stereosubtypes: *VectorTileDataService* for serving vector tiles, *RasterTileDataService* for serving raster tiles, *VectorDataService* for serving vector datasets, and *RasterDataService* for raster datasets. These stereosubtypes are addressed separately because each service represents different ways of serving spatial data.

VectorTileDataService Class Stereotype

- This class stereotype represents a service that provides access to vector tiles, a format of storing geographic data with efficient rendering capabilities at different zoom levels.
- *relevantLayers* attribute: an array of strings that lists the collection of layer names that are served by the vector tile service, since the data service might serve several vector tiles.
- This class stereotype inherits attributes and operations from the *DataService* and *SpatialDataService* class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

RasterTileDataService Class Stereotype

- This class stereotype represents services that provide access to raster tiles, which are smaller images that can be combined to form a larger image.
- This class stereotype inherits attributes and operations from the *DataService* and *SpatialDataService* class stereotypes.
- This class stereotype does not have public attributes or operations to be used or inherited by other class stereotypes.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

VectorDataService Class Stereotype

- This class stereotype represents data services that provides access to vector data.
- *geometry* attribute: a string that specified geometry type defined by ISO 19107:2003.
- *relevantAttributes* attribute: an array of strings referring to data properties that have meaningful role in the GWA based on the user preference.
- This class stereotype inherits available attributes and operations from *DataService* and *SpatialDataService* class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

RasterDataService Class Stereotype

- This class stereotype represents data services that provides access to raster data.
- *imageBounds* attribute: an array of coordinates referring to the geographic extent of the raster data.
- *pixelSize* attribute: an array of integer representing the size of a single pixel in the raster data. This attribute has a tagged value of *unit* to indicate the unit of the size.
- *pixelAttributeDataType* attribute: a string of the defined data type used for the pixel value.
- This class stereotype inherits available attributes and operations from *DataService* and *SpatialDataService* class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

Below is the elaboration for the association stereotypes in this profile.

Offer Association Stereotype

- This association stereotype represents the movement of data from one entity to another in a data modelling context.
- It is used to associate instances of the *DataStore* or *DataService* class stereotype or the stereotypes that inherit them, with instances of *Dataset* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it captures the relationship where the source offers available datasets.

Display Association Stereotype

- This association stereotype represents a visualization relationship between data and its graphical representation on a map in some GWA.
- This association stereotype is used to associate instances of the *Dataset* or *DataService* class stereotype or the stereotypes that inherit them, with instances of the *MapDisplay* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it clearly conveys the purpose of the association where data is being displayed by a map in some GWA.

Process Association Stereotype

- This association stereotype represents data handling before the data is being served by WHAT.
- It is used to associate instances of the *Dataset* or *DataService* class stereotype or the stereotypes that inherit it, with instances of *PimControllerCore* profile's class stereotypes. This is semantically appropriate because controller classes may act as the central processing units in a GWA that perform data manipulation, analysis, and transformation.

Bind Association Stereotype

- This association stereotype represents the reliance of a chart component on some data.
- It is used to associate instances of the *Dataset* or *DataService* class stereotype or the stereotypes that inherit them, with instances of *ChartDisplay* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it captures the data dependency of the visual component on the provided data.

5.1.2. PIM MVC-View Profile

Figure 5.5 shows the contents of PimViewCore profile which handles visualization-related concepts. The following paragraphs describe each stereotype in detail.

Below is the elaboration for UiComponent class stereotypes and its stereosubtypes.

UiComponent Class Stereotype

- This class stereotype represents the foundational building block for user interface (UI) components in a GWA.
- This class stereotype does not inherit nor does it have public attributes or operations to be inherited by other class stereotypes.
- This class stereotype has inheritance relationships with four stereosubtypes: *DataVisualization* for representing data on the interface, *LayerSelection* for representing options for the user to choose, Button for UI component dedicated for triggering actions, and *SearchBar* for user input focused on searching data. These stereosubtypes are addressed separately because they provide specialized functions beyond the basic UI components.

DataVisualization Class Stereotype

- This class stereotype represents UI components specialized in creating visual elements that communicate data insights.
- *VisualizationTitle* attribute: a string that describes the visualization.
- It inherits attributes and operations from the *UiComponent* class stereotype.
- It has inheritance relationships with two stereosubtypes: *MapDisplay* for displaying map for spatial data and *ChartDisplay* for displaying charts. These stereosubtypes are addressed separately because they cater for different types of visualization with different configurations and purposes.

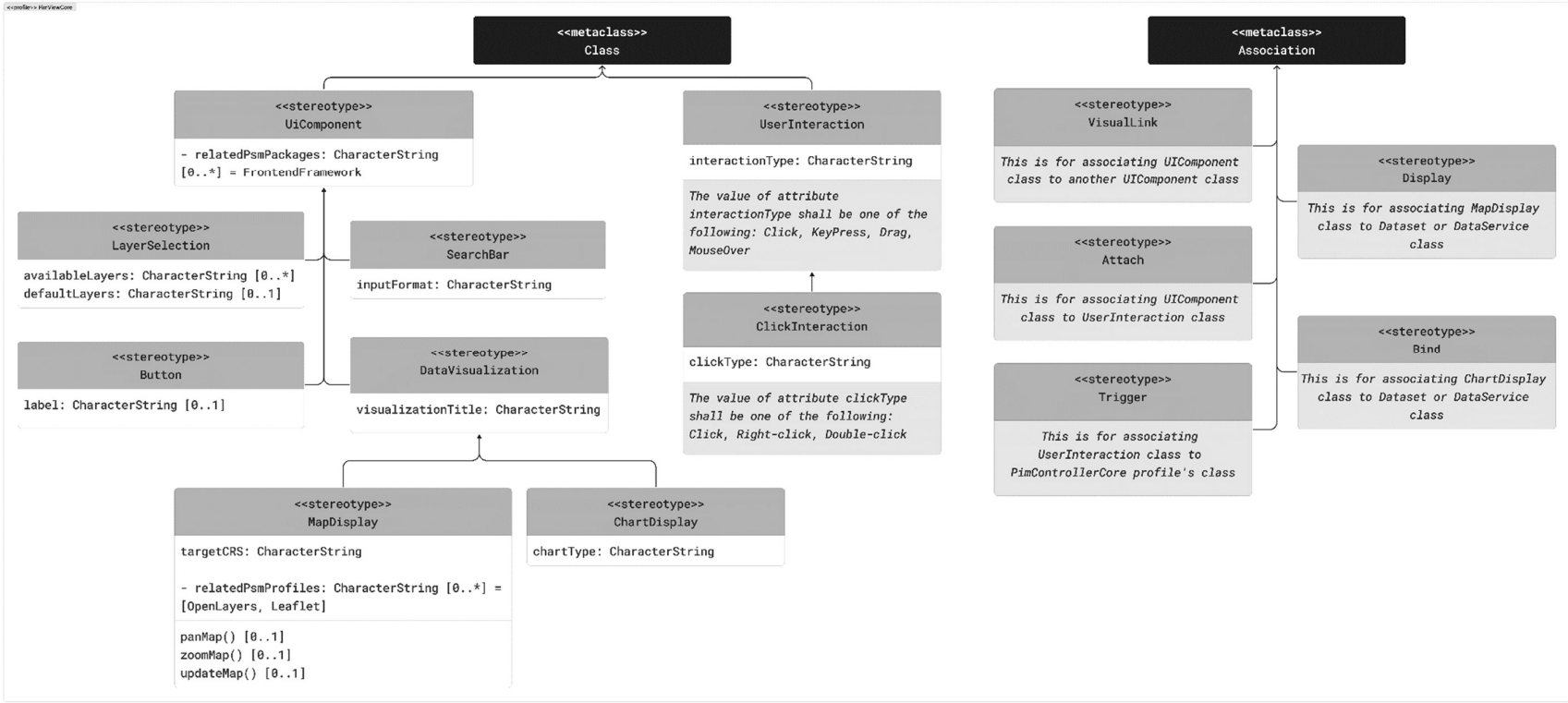


Figure 5. 5 The PimViewCore profile stereotypes

MapDisplay Class Stereotype

- This class stereotype represents maps for visualizing geospatial data on a user interface.
- *targetCRS* attribute: a string that determines the CRS the map is using. This information is useful to indicate a potential coordinate transformation for the used data.
- *panMap* operation: a function to shift the map in specific direction.
- *zoomMap* operation: a function to adjust the zoom level of the map.
- *updateMap* operation: a function to refresh the map display with new or updated data.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

ChartDisplay Class Stereotype

- This class stereotype represents visual data representation through charts.
- *ChartType* attribute: a string that specifies the type of chart.
- This class stereotype inherits attributes and operations from the *UiComponent* and *DataVisualization* class stereotypes.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

LayerSelection Class Stereotype

- This class stereotype represents a UI component that contains the layer options for a user to choose from, such as UI component with list of available data.
- *availableLayers* attribute: an array of strings that lists all possible layer options that the user can choose from.
- *defaultLayers* attribute: a string representing the default layers that are selected when the UI component loads.
- This class stereotype inherits available attributes and operations from the *UiComponent* class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

Button Class Stereotype

- This class stereotype represents a clickable UI element that triggers an action.
- *label* attribute: a string that is the text label displayed on the button.
- This class stereotype inherits attributes and operations from the UiComponent class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

SearchBar Class Stereotype

- This class stereotype represents a UI component specifically designed for users to enter search queries.
- *inputFormat* attribute: a string that represents the format or type of the data the search bar can accept.
- This class stereotype inherits available attributes and operations from the UiComponent class stereotype.
- This class stereotype does not have inheritance relationships with other stereosubtypes.

Below is the elaboration for UserInteraction class stereotypes and its stereosubtypes. UserInteraction is different from UiComponent because user interaction refers to the interaction that happen to UI component, which is a separate concept from the UI component itself.

UserInteraction Class Stereotype

- This class stereotype represents general user interaction elements in a GWA.
- *interactionType* attribute: a string that specifies the type of user interaction. The value of this attribute shall be one of the following: *Click*, *KeyPress*, *Drag*, or *MouseOver*.
- *executeCallbackFunction* operation: a function that executes the callback function that is triggered after a user interaction occurs.
- This class stereotype does not inherit attributes or operations from other class stereotypes.
- This stereotype has an inheritance relationship with a class stereosubtype: *click* for representing an action that creates click event.

ClickInteraction Class Stereotype

- This class stereotype represents the action of clicking an UI component.
- *clickType* attribute: a string that specifies the type of click. The value of this attribute shall be one of the following: *Click*, *Right-click*, and *Double-click*.
- This class stereotype inherits available attributes and operations from *UserInteraction* class stereotype.
- This class stereotype does not have inheritance relationships with more specific stereotypes.

Below is the elaboration for the association stereotypes in this profile.

VisualLink Association Stereotype

- This association stereotype represents the way a UI component links with other UI components.
- This association stereotype is used to associate instances of *UiComponent* class stereotype or the stereotypes that inherit it, with instances of *UiComponent* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it describes the way UI components interact and influence each other visually on the screen.

Attach Association Stereotype

- This association stereotype represents the way a UI component captures and transmit user input into an element responsible for handing the user interaction.
- This association stereotype is used to associate instances of *UiComponent* class stereotype or the stereotypes that inherit it, with instances of *UserInteraction* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it describes the flow of user interaction data.

Trigger Association Stereotype

- This association stereotype represents the way a user interaction triggers a function in GWA.
- This association stereotype is used to associate instances of *UserInteraction* class stereotype or the stereotypes that inherit it, with another instance of *PimControllerCore* profile's class stereotypes. This is semantically appropriate because it is a common semantic for the user interaction event to trigger a function.

Display Association Stereotype

- This association stereotype represents a visualization relationship between data and its graphical representation on a map in GWA.
- This association stereotype is used to associate instances of *Dataset* or *DataService* class stereotype or the stereotypes that inherit them, with instances of *MapDisplay* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it clearly conveys the purpose of the association where data is being displayed by a map in GWA.

Bind Association Stereotype

- This association stereotype represents the reliance of a chart component for data.
- This association stereotype is used to associate instances of *Dataset* or *DataService* class stereotype or the stereotypes that inherit it, with instances of *ChartDisplay* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it captures the data dependency of the visual component on the provided data.

5.1.3. PIM MVC-Controller Profile

Figure 5.6 shows the contents of PimControllerCore profile which handles visualization-related concepts. The following paragraphs describe each stereotype in detail.

DataProcessing Class Stereotype

- This class stereotype represents components responsible for manipulating and transforming data.
- *outputFormat* attribute: a string that specifies the format of the data after processing.
- *supportedProcesses* attribute: an array of string referring to the processes that can be performed.
- *executeProcess* operation: a function that defines the execution of the process.
- This class stereotype does not inherit any attribute and operation from other class stereotypes.
- This class stereotype has inheritance relationships with two specific stereotypes: *SpatialProcessing* for handling spatial data processing and *NonSpatialProcessing* for handling non-spatial data processing. These specific stereotypes are addressed separately because both provides different means to handle different data types and provides specialized functionalities.

SpatialProcessing Class Stereotype

- This class stereotype represents processes that operate on spatial data.
- *outputCRS* attribute: a string that specify the CRS of the output data after process.
- This class stereotype inherits available attributes and operations from *DataProcessing* class stereotype or the stereotypes that inherit it.
- This class stereotype has inheritance relationships with two specific stereotypes: *LocalProcessing* for processes sourced from local machine and *CloudProcessing* for processes sourced from cloud provider. These specific stereotypes are addressed separately because the environment and resources are both different and require different ways of handling the processes.

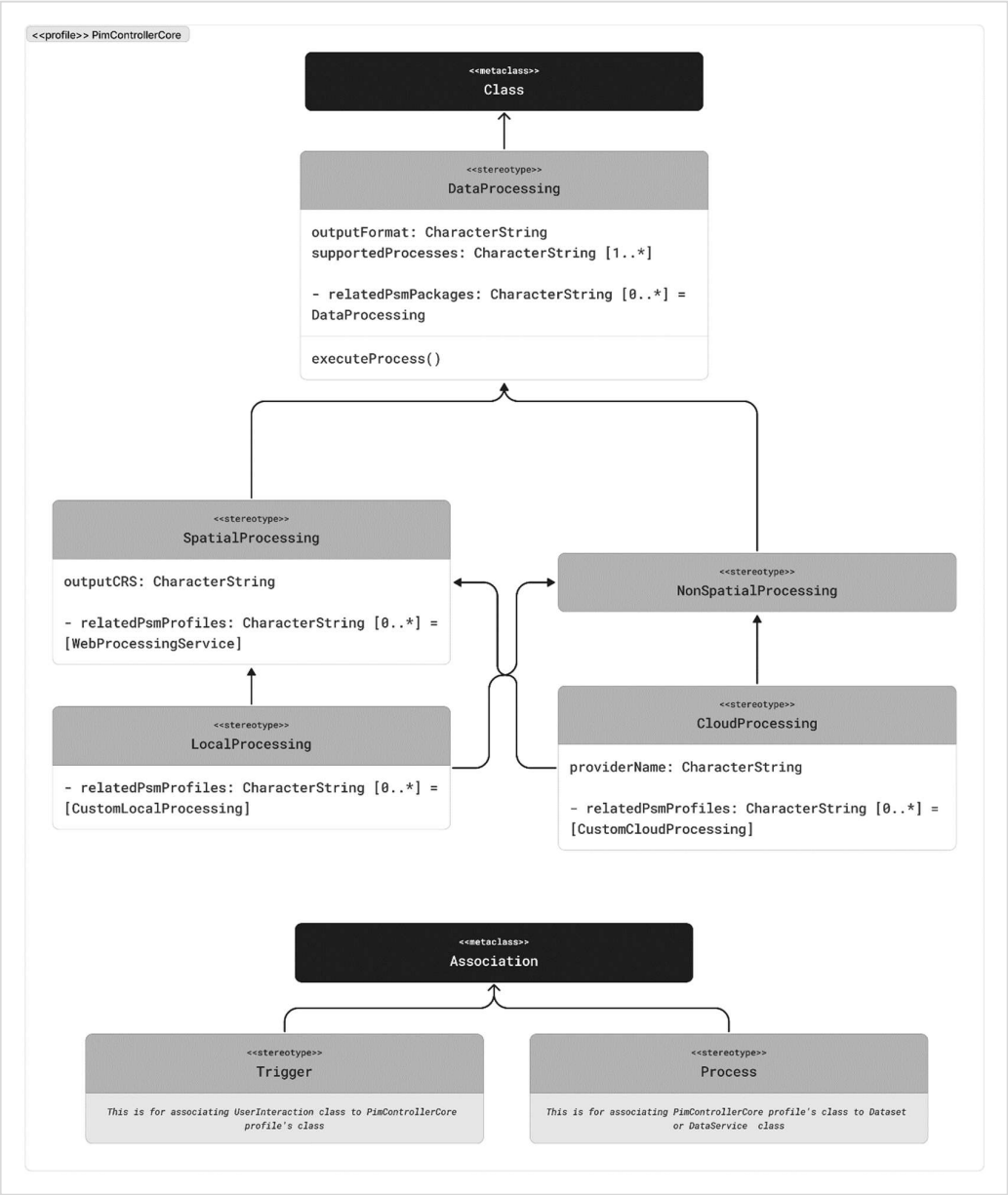


Figure 5. 6 The PimControllerCore profile stereotypes

NonSpatialProcessing Class Stereotype

- This class stereotype represents processes that do not involve spatial data.
- This class stereotype does not have public attribute and operation to be inherited by other class stereotypes.
- This class stereotype inherits available attributes and operations from *DataProcessing* class stereotype.
- This class stereotype has inheritance relationships with two specific stereotypes: *LocalProcessing* for processes sourced from local machine and *CloudProcessing* for processes sourced from cloud provider. These specific stereotypes are addressed separately because the environment and resources are both different and require different ways of handling the processes.

LocalProcessing Class Stereotype

- This class stereotype represents processes that are accessed and handled in the local machine.
- This class stereotype does not have public attribute and operation to be inherited by other class stereotypes.
- This class stereotype inherits available attributes and operations from *DataProcessing*, *SpatialProcessing*, and *NonSpatialProcessing* class stereotypes.
- This class stereotype does not have inheritance relationships with more specific stereotypes.

CloudProcessing Class Stereotype

- This class stereotype represents processes that are accessed and handled within cloud environment.
- This class stereotype does not have public attribute and operation to be inherited by other class stereotypes.
- This class stereotype inherits available attributes and operations from *DataProcessing*, *SpatialProcessing*, and *NonSpatialProcessing* class stereotypes.
- This class stereotype does not have inheritance relationships with more specific stereotypes.

Trigger Association Stereotype

- This association stereotype represents the way a user interaction triggers a function in GWA.
- This association stereotype is used to associate instances of *UserInteraction* class stereotype or the stereotypes that inherit it, with another instance of *PimControllerCore* profile's class stereotypes. This is semantically appropriate because it is a common semantic for the user interaction event to trigger a function.

Process Association Stereotype

- This association stereotype represents a relationship that represents the way processes handles data input to generate the output.
- This association stereotype is used to associate instances of *PimControllerCore* profile's class stereotype with instances of Dataset or *DataService* class stereotype or the stereotypes that inherit it. This is semantically appropriate because it defines the way processes handle datasets from any sources including data services.

5.2. PSM Profile Development

This step developed the possible UML profiles containing stereotypes to be used in building a Platform Specific Model (PSM), or as it is named PSM-Profiles. The step started with designing how the PSM-Profiles are structured with PIM transformation in mind. This consideration is required because every information needed to develop a stereotype in PSM-Profiles must come from existing PIM-Profiles' stereotypes in any way. This allows the instance of PIM to be transformed into PSM with clear logic behind it.

The most suitable target users to reproduce this step are the User Type #2: the Developers. By knowing how to build the PSM-Profiles, users can reproduce the step to build their own UML profile for building the PSM or to improve the existing ones.

Figure 5.7 shows the UML package diagram that showcases the structure of the PSM-Profiles. The top rectangle *Psm* represents a package for the PSM itself while the apply arrows represent the direct

relationship that showcase which packages are being applied. The first level of packages displays the three packages that result from the implementation of the MVC separation of concerns. This separation gives a boundary for the transformation of each PIM-Profiles stereotype as it can only be transformed into PSM-Profile stereotypes that come from the same MVC layer.

Each MVC package is separated further into several purpose-related packages which relate to a specific collection of available technologies:

1. PsmModel package contains DataStore package representing the data storage technologies such as PostgreSQL, and DataService package representing services that provide data access such as WMS and WFS.
2. PsmView package contains FrontendFramework representing user interface libraries such as React, and MappingLibrary package representing web mapping libraries such as Leaflet.
3. PsmController package contains BackendFramework representing web server framework such as Express.JS, HttpClientLibrary representing HTTP client libraries such as Axios, and DataProcessing package representing geospatial processing services such as WPS.

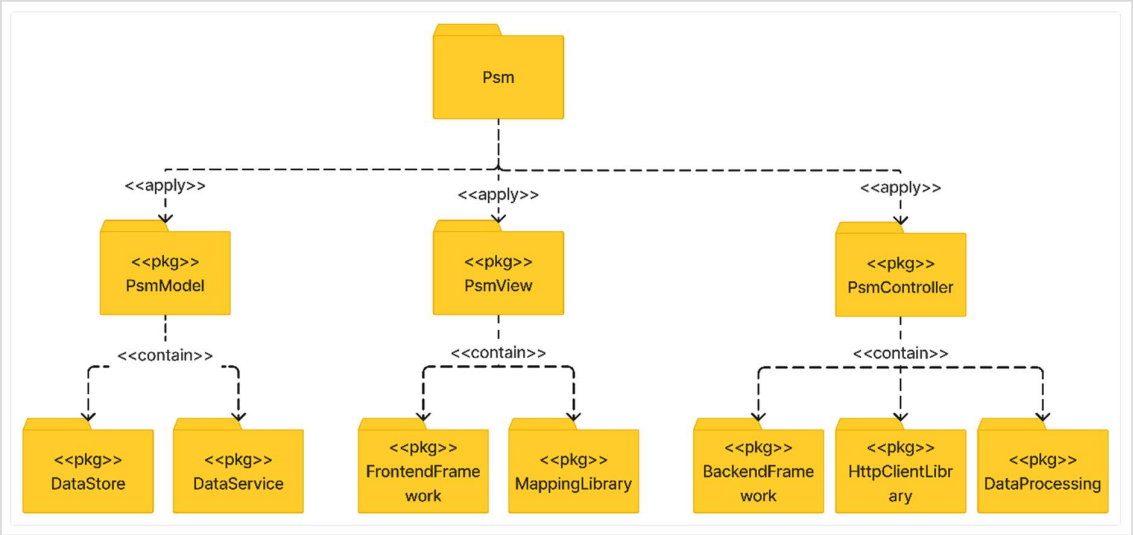


Figure 5. 7 The structure of the proposed PSM-Profiles packages based on MVC design pattern.

Figure 5.8 shows the proposed PSM-Profiles of each technology in this research. The colour yellow indicates the PSM-Profile used in this research project. Not all PSM-Profiles were elaborated in detail because the platform-specificity of the contained stereotypes might add too much irrelevance to the project. PSM-Profiles are demonstrated in the MDA Implementation chapter.

Figure 5. 8 The proposed PSM-Profiles contained in each package.

5.3. PIM Instantiation

This section explains how to use PIM-Profiles to build a PIM instantiation. This step was performed with reproducibility in mind. The most suitable target users to reproduce this step are:

- User Type #1: the Designers.
- User Type #2: the Developers.

By knowing how to instantiate a PIM from the developed PIM-Profiles in this section, both users can reproduce the step to build their own preferred PIM. PIM instantiation is part of model transformation development step, which uses the PIM-Profiles that were developed in the UML profile development as the only provided resource. The result of this section is the steps to perform PIM instantiation, which can be used to create a PIM. Figure 5.9 shows the parts of research flowchart that are relevant to this section.

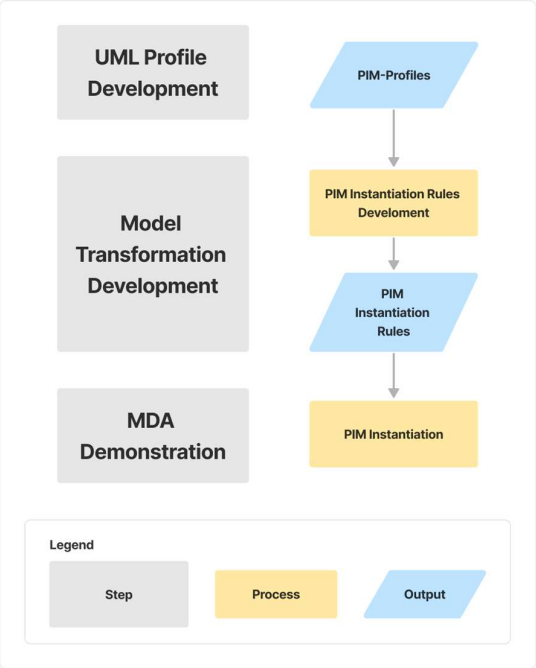


Figure 5. 9 Part of Research Flowchart that are related to PIM Instantiation

Several requirements were developed to help users complete the PIM instantiation according to the proposed approach. Following the requirements below in sequence allows users to create a PIM diagram that can be brought into model transformation into PSM. Figure 5.10 shows the overview of the requirements used for PIM instantiation.

5.3.1. Class Instantiation

Recommendation #5.3.1

To design an instance of PIM class, it is recommended to use child stereotypes of the PIM-Profiles.

Resource: PIM-Profiles (provided); Execute: Manual by User

The first step for users is to find the available PIM-Profiles stereotypes to use. To direct the user's attention to the most valuable stereotypes to use, it is recommended to use the child classes which is shown as the lower-level classes in the diagram structure. This recommendation is highly suggested as child classes contain more information including attributes and operations that are inherited by their parent classes. This makes child classes more informative to use for designing the PIM as well as its transformation in later steps.

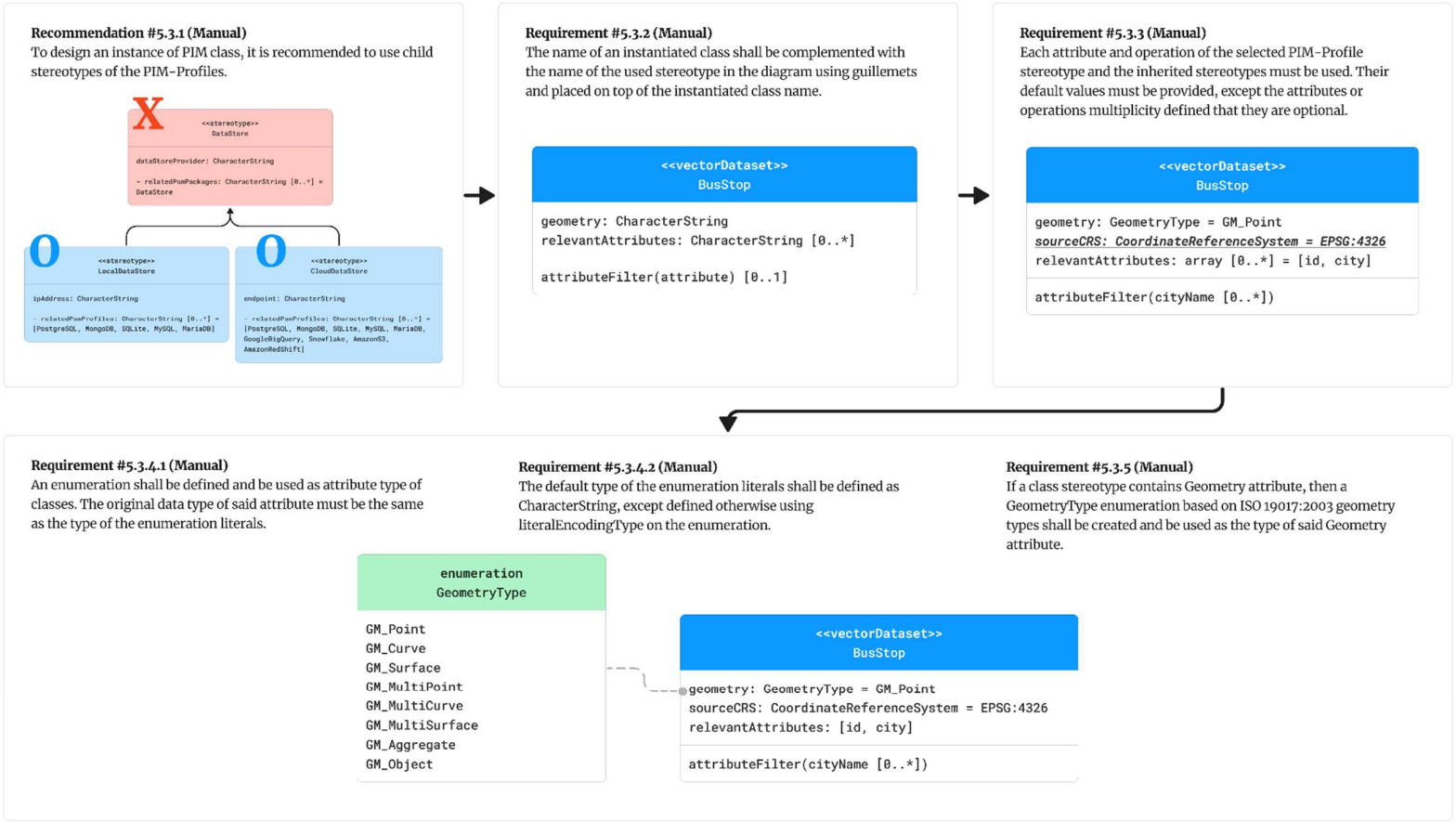


Figure 5. 10 Overview of requirements for PIM instantiation (part 1)

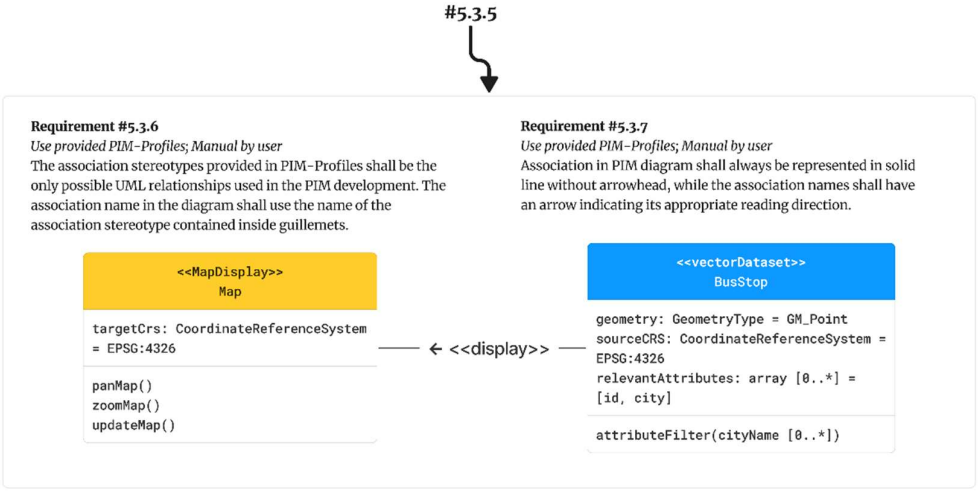


Figure 5. 11 Overview of requirements for PIM instantiation (part 2)

This recommendation also considers that the model transformation scripts might have mandatory information to be filled to perform the automatic transformation. The use of parent class leads to fewer information to be received by the model transformation script. This can potentially cause the transformation step to give more prompt users to manually fill in for the empty required information (such as the way to access the data store in the given example) which makes the transformation less automated. This issue might also introduce conflicts such as increased chance of incompatibility with other selected PSM technologies or even failure which might stop the transformation process.

Figure 5.12 shows the example of *DataStore* class stereotype which has inheritance relationship with two child class stereotypes: *LocalDataStore* and *CloudDataStore*. Recommendation #5.3.1 suggests on using the two child class stereotypes instead of the *DataStore* class stereotype which has fewer information.

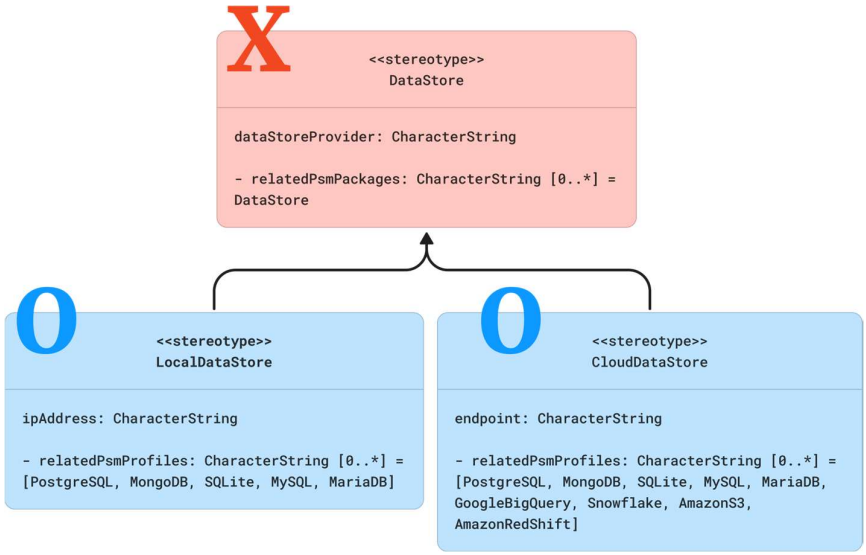


Figure 5. 12 Selecting the child classes of DataStore class stereotypes.

5.3.2. Class Name

Requirement #5.3.2
The name of an instantiated class shall be complemented with the name of the used stereotype in the diagram using guillemets and placed on top of the instantiated class name.

Resource: PIM-Profiles (provided); Execute: Manual by User

After selecting the preferred PIM-Profiles stereotype to use, users are required to give name of the instantiation of the selected class stereotype. In the instantiated class stereotype, the name of the stereotype becomes a keyword in guillemets (« ») and placed on top of the instantiated class name. Figure 5.13 shows example of *VectorDataset* stereotype being used to instantiate *BusStop* class.

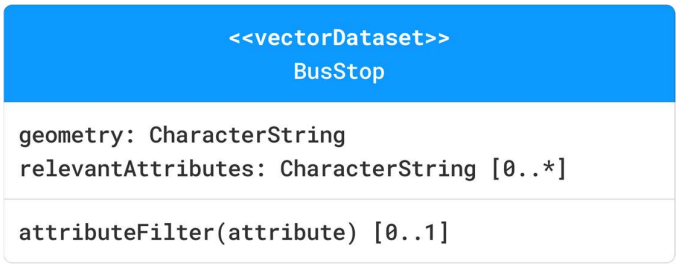


Figure 5. 13 Example of PIM class name convention

5.3.3. Attributes and Operations

Requirement #5.3.3
Each attribute and operation of the selected PIM-Profiles stereotype and the inherited stereotypes must be used. Their default values must be provided, except the attributes or operations multiplicity defined that they are optional.

Resource: PIM-Profiles (provided); Execute: Manual by User

After a new class is created, users are required to provide the default values of each available attribute and operation of the selected stereotype as well as all inherited ones from its parent stereotypes. All attributes and operators have the default multiplicity of [1..1] meaning that it is mandatory to be provided, except it is defined otherwise with other multiplicity. The attribute and operations that have multiplicity configurations starting with 0 such as [0...1] and [0..*] indicate that the attribute or operation is optional. Figure 5.14 shows example of how the *DataService*, *SpatialDataService*, and VectorDataservice class stereotypes are used to instantiate *CityBoundary* class in the PIM.

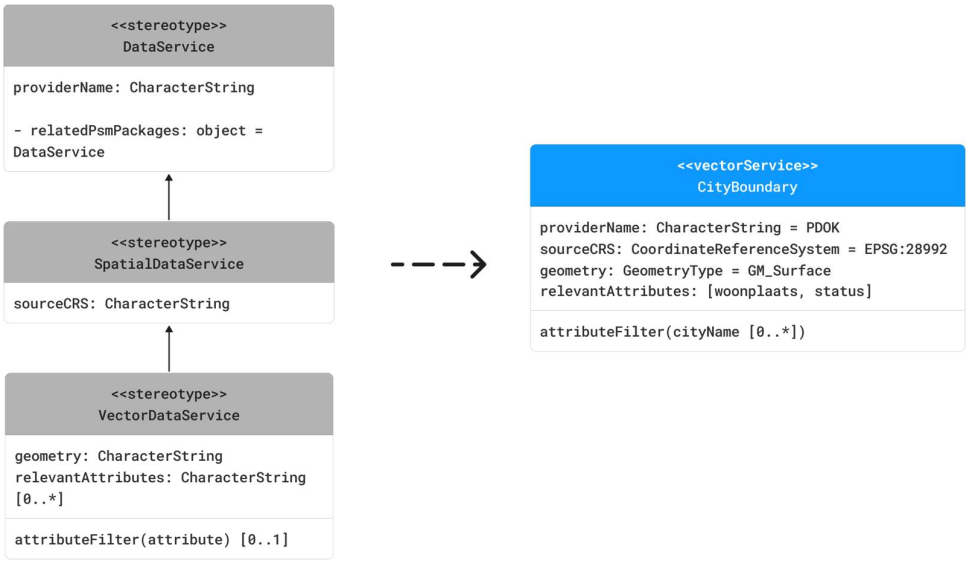


Figure 5. 14 Example of inherited class attributes and operations in PIM

5.3.4. Enumeration

Requirement #5.3.4.1
An enumeration shall be defined and be used as attribute type of classes. The original data type of said attribute must be the same as the type of the enumeration literals.

Resource: None; Execute: Manual by User

Requirement #5.3.4.2
The default type of the enumeration literals shall be defined as CharacterString, except defined otherwise using literalEncodingType on the enumeration.

Resource: None; Execute: Manual by User

Other than class and association stereotypes, the UML generic concept of enumeration can also be used in the PIM. Enumerations in the PIM can be use as type of a class attribute. In this case, there is no need for the class-enumeration relationship to be indicated using association, and the original type data type of said attribute must be the same as the type of enumeration literals. The default value of the attribute comes from the enumeration literals. Figure 5.15 shows how the *BusStop* class uses several enumerations for its attributes. Figure 5.16 shows an example of how *literalEncodingType* works in an enumeration.

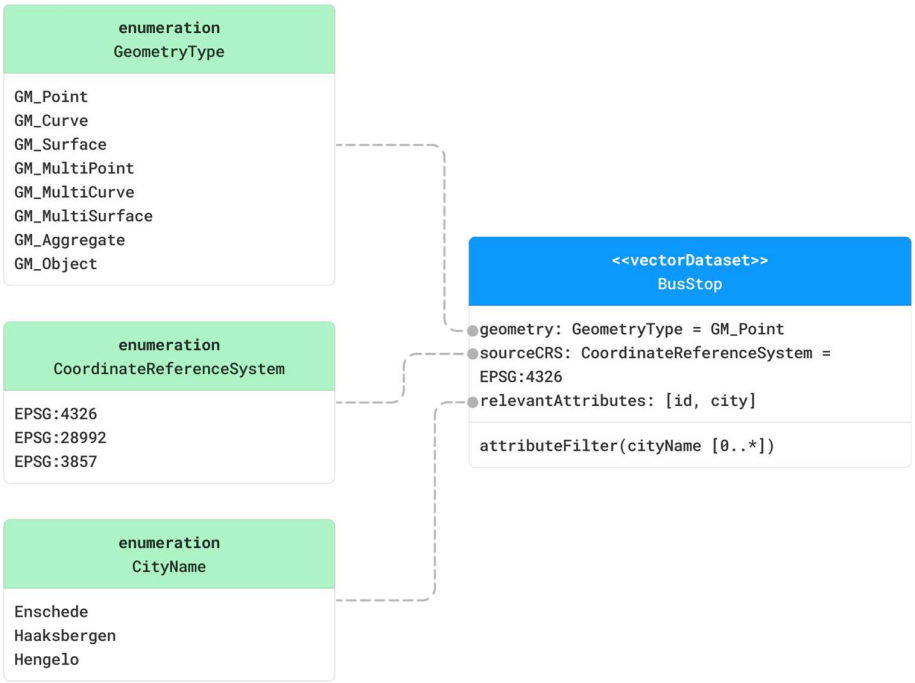


Figure 5. 15 Examples of the use of enumeration as attribute type

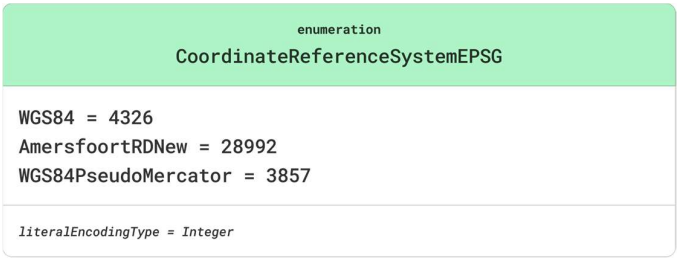


Figure 5. 16 Examples of the use of *literalEncodingType* in enumeration

5.3.5. Enumeration – Geometry Type

Requirement #5.3.5
If a class stereotype contains Geometry attribute, then a GeometryType enumeration based on ISO 19017:2003 geometry types shall be created and be used as the type of said Geometry attribute.

Resource: None; Execute: Manual by User

In addition to user-defined enumerations, a GeometryType enumeration must be created when an instance of class stereotypes that contain Geometry attribute is used in the PIM. The literals of GeometryType enumeration come from the ISO 19017:2003 geometry types. This enumeration becomes the attribute type of each Geometry attributes in the PIM.

In the current PIM-Profiles developed in this research, these classes are VectorDataset and VectorDataService. The purpose of this obligation is to use standardized set of geometry types that is compatible with JSON schema, one of main definitions in this research. This standardized approach is proven to be useful in PIM to PSM transformation where the compatibility with JSON schema allows the conversion of the developed PIM into JSON. Figure 5.17 shows the GeometryType enumeration in the PIM of this research.

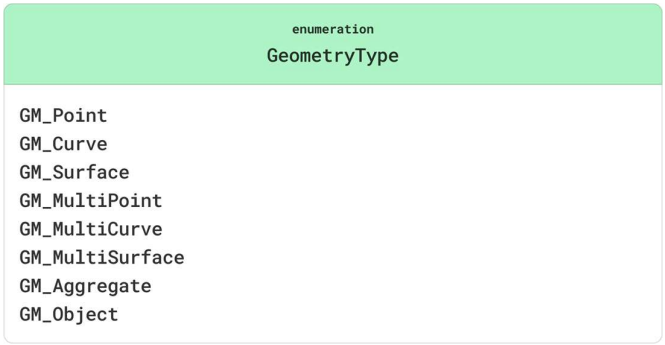


Figure 5. 17 Example of enumeration in PIM for GeometryType

5.3.6. Association

Requirement #5.3.6

The association stereotypes provided in PIM-Profiles shall be the only possible UML relationships used in the PIM development. The association name in the diagram shall use the name of the association stereotype contained inside guillemets.

Resource: PIM-Profiles (provided); Execute: Manual by User

After all classes are instantiated in the PIM, associations are required to define the relationship between each class. It is up to the user decision whether a class has association relationship. If it is decided that a class have an association, the available association stereotypes contained in PIM-Profiles are the only options to be used in the PIM.

Each association stereotypes have limited use as described in the PIM-Profiles and are developed to be as semantically appropriate as possible. For example, display can only be used to associate a class stereotype from PimModelCore profile with MapDisplay class from PimViewCore profile, since it is appropriate to state that a Map 'display' a Dataset. The name of the association stereotype contained in guillemets (« ») becomes the association name.

5.3.7. Association Annotation

Requirement #5.3.7
Association in PIM diagram shall always be represented in solid line without arrowhead, while the association names shall have an arrow indicating its appropriate reading direction.

Resource: PIM-Profiles (provided); Execute: Manual by User

The association in PIM shall always be a bi-directional association, shown with a solid line. This means that both class in an association can refer to each other. In addition to the directionality of the association, the association stereotype names are also configured using single verbs that it only makes sense to be read in certain direction. This particular trait is indicated by the order the association stereotype is described in the PIM-Profiles. For example, the way display association stereotype is described: *"… for associating MapDisplay class to Dataset or DataService class"* indicates that it only makes sense to state that a Map 'display' a Dataset, not the other way around. This configuration is useful during PIM-to-PSM transformation later. In PIM diagram, this trait is shown using an arrow next to the association name. Figure 5.18 shows example of how the Display and Offer association stereotypes in PimModelCore profile are being used to associate the described class stereotypes.

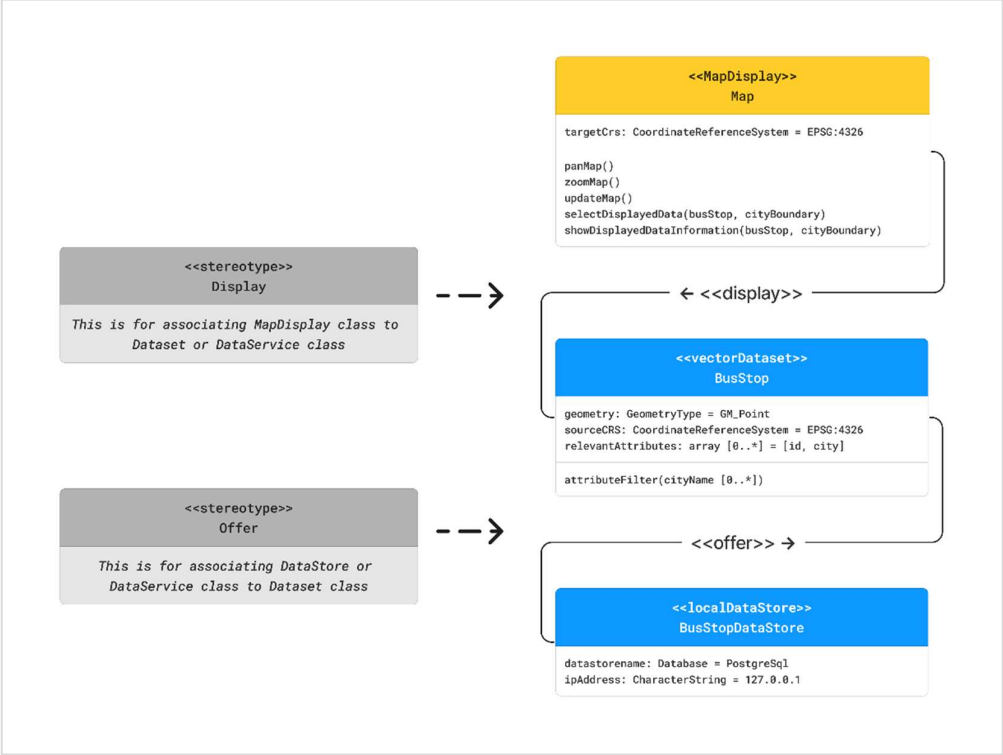


Figure 5. 18 Examples of association instance in PIM

5.4. PIM to PSM Transformation Rules

This section explains about how to create a model transformation of PIM into PSMs. This step was performed with reproducibility in mind. The most suitable target users to reproduce this step are the User Type #2: the Developers. By knowing how to build the model transformation users can reproduce the step to transform their PIM into PSM using their selected technology. The rest of this section elaborates on all steps and considerations in developing the model transformation.

5.4.1. UML to JSON Conversion

The first step in the transformation process is to convert the designed PIM diagram into JSON format. As previously explained in Research Method chapter, JSON conversion was performed to allow automated model transformation of the designed diagram. The following subsections explain the requirements of this step in detail. Each requirement will be complemented with information about what resource and data being used in the requirement, whether the resource is being provided from the beginning or the resource is something that is result from previous steps within this project, as well as whether the requirement is something that the user need to do manually or it is a part of the automated transformation. Figure 5.19 shows the parts of research flowchart that are relevant to this section.

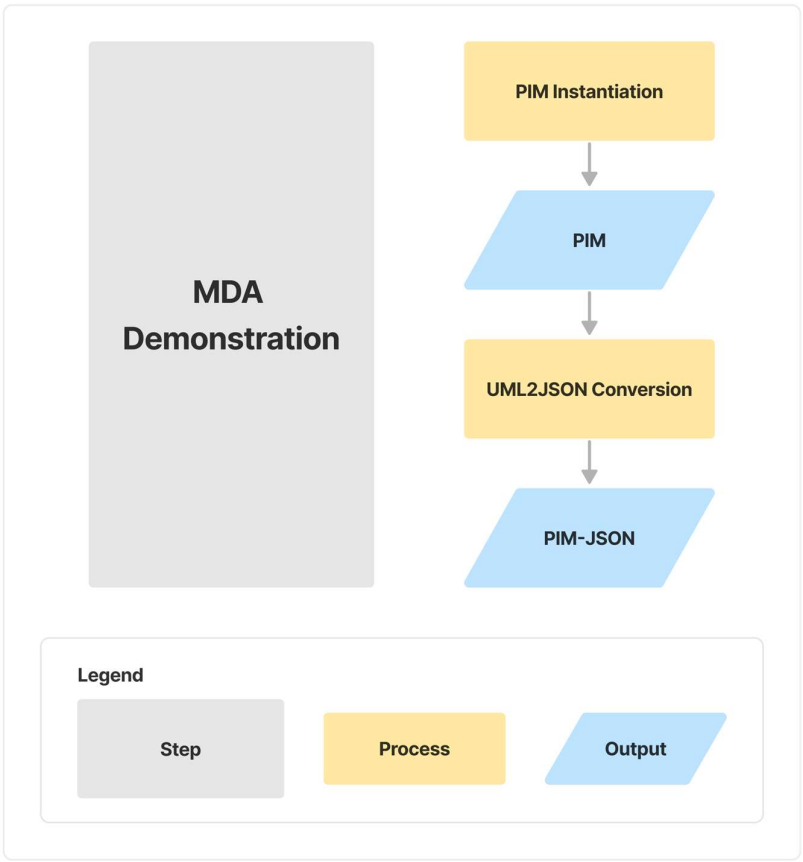


Figure 5. 19 Part of Research Flowchart that are related to UML to JSON conversion.

Several requirements were developed to help users complete the UML to JSON conversion according to the proposed approach. Following the requirements below in sequence allows users to convert the PIM diagram into JSON schema before the transformation to PSM takes place. Figure 5.20 shows the summary of the requirements used for the conversion.

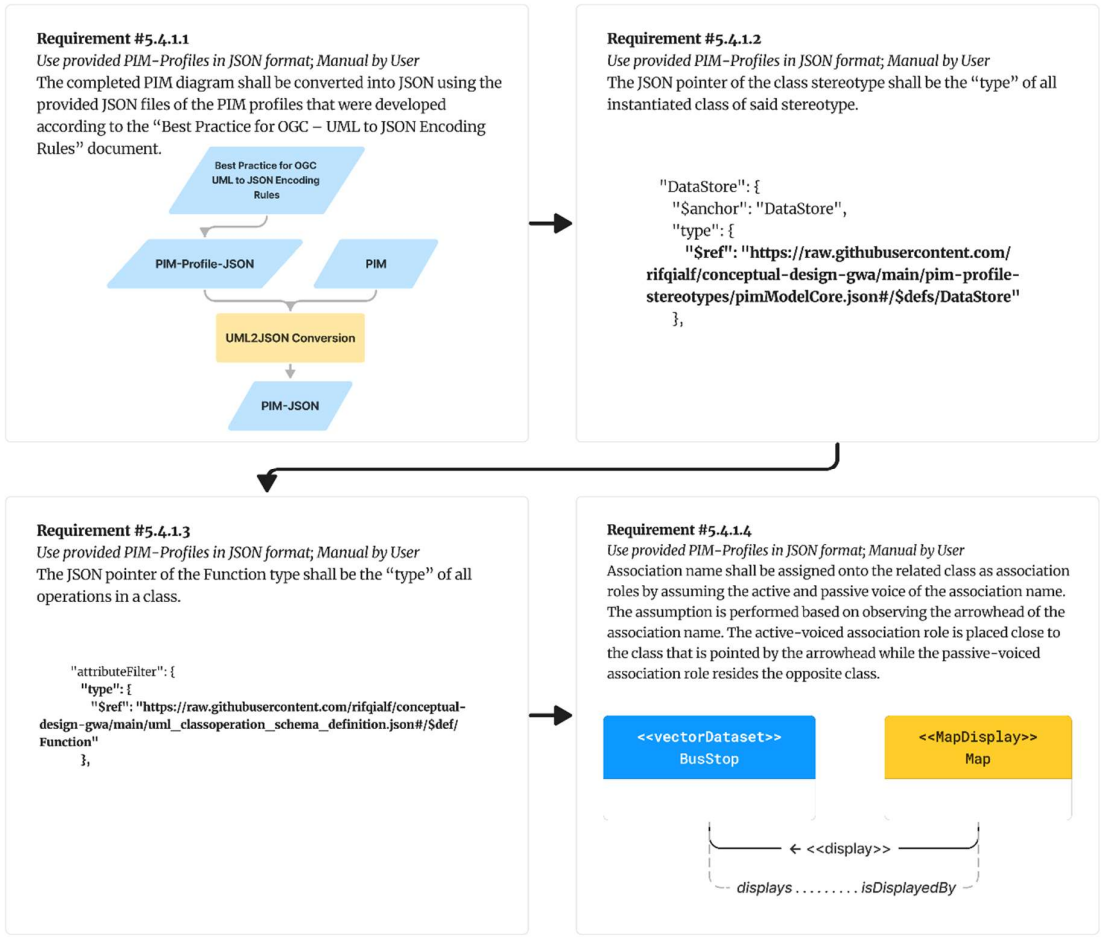


Figure 5. 20 Overview of requirements for UML to JSON conversion.

5.4.1.1. PIM-Profiles JSON Schema

Requirement #5.4.1.1
The completed PIM diagram shall be converted into JSON using the provided JSON files of the PIM profiles that were developed according to the "Best Practice for OGC – UML to JSON Encoding Rules" document.

Resource: PIM-Profiles in JSON format (provided); Execute: Manual by User

As previously mentioned, his conversion process is a manual process by following "Best Practice for OGC – UML to JSON Encoding Rules" (OGC, n.d.-a) or the UML2JSON best practice document for short. To make the conversion process convenient for users, JSON templates of PIM-Profiles were developed based on the best practice document. Figure 5.21 shows sample of PIM-Profiles in JSON format.

This guided conversion is important to allow the designed diagram to be stored in a transformable format while also retaining all the information. This research does not elaborate the detailed steps in transforming PIM diagram into PIM-JSON and suggests users to follow the best practice document. In addition to the guided conversion above, several extra steps were required for the conversion in this research such as JSON schema for stereotypes, functions, and association roles.

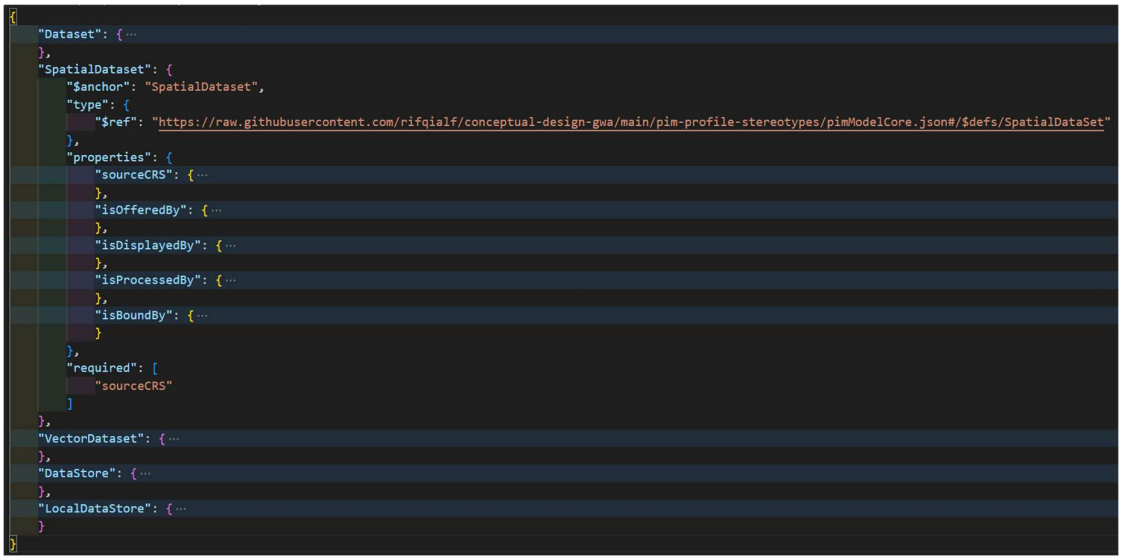


Figure 5. 21 Example of PIM-Profiles in JSON schema

5.4.1.2. JSON Schema for Stereotypes

Requirement #5.4.1.2
The JSON pointer of the class stereotype shall be the "type" of all instantiated class of said stereotype.

Resource: PIM-Profiles in JSON format (provided); Execute: Manual by User

The current UML2JSON best practice document does not specifically include stereotype into the encoding. However, the detail about which class stereotype that is being used to instantiate a class is

relevant to the model transformation in later step. To solve this issue, the research proposes that the type of each class shall refers to the JSON pointer of the class stereotype's external schema. This approach is stated in the UML2JSON best practice document suggestion regarding the provision of external types.

This approach requires the PIM-Profiles to be converted into JSON schema and store the schema online to be referred. Following the UML2JSON best practice document's Requirement 2 about the way to reference types from the external schema, PIM-Profiles' stereotypes can be referred by each relevant class as the class type. Figure 5.22 shows the example of *BusStopDataStore* class in PIM that refers *LocalDataStore* class stereotype as the class type.

```
$anchor: "BusStopDataStore",
type: {
  $ref: "https://raw.githubusercontent.com/rifqialf/conceptual-design-gwa/main/pim-profile-stereotypes/pimModelCore.json#/$defs/LocalDataStore",
},
properties: {
  datastorename: {
    type: {
      $ref: "#/$defs/Database",
    },
    default: "postgresql",
  },
  location: {
    type: "string",
    default: "localhost"
```

Figure 5. 22 Example of PIM stereotype JSON pointer

5.4.1.3. JSON Schema Functions

Requirement #5.4.1.3
The JSON pointer of the Function type shall be the "type" of all operations in a class.

Resource: PIM-Profiles in JSON format (provided); Execute: Manual by User

The current UML2JSON best practice document only covers the way attributes can be converted into JSON encoding, but not operations. According to the UML2JSON best practice document, attributes are represented as "property" which has the definition of *"attribute or association role; not of an enumeration or code list"*. Assuming from this flexible definition, this research decided to store operations as part of property together with attributes and association roles. However, there is also a need to make operations to be distinctly structured in the JSON schema, so the model transformation script can split the differences between attributes, association roles, and operations.

To solve this issue, this research proposed an external JSON schema for describing operations in the JSON encoding in this research, shown in Figure 5.23. This approach is stated in the UML2JSON best practice document suggestion regarding the provision of external types. This external schema allows operations of UML classes to be identified as the type "Function" and distinct themselves among attributes and association roles.

```
{
    "$schema": "https://json-schema.org/draft/2019-09/schema",
    "$id": "https://raw.githubusercontent.com/rifqialf/conceptual-design-gwa/main/uml_classoperation_schema_definition.json",
    "$defs": {
        "Function": {
            "$anchor": "Function",
            "title": "function",
            "description": "definition of a function to represent UML class operation",
            "type": "object",
            "required": [
                "href"
            ],
            "properties": {
                "href": {
                    "type": "string",
                    "description": "Supplies the URI to a remote resource (or resource fragment).",
                    "example": "http://data.example.com/buildings/123"
                },
                "rel": {
                    "type": "string",
                    "description": "The type or semantics of the relation.",
                    "example": "related"
                },
                "title": {
                    "type": "string",
                    "description": "Used to label the name of the function.",
                    "example": "AttributeFilter"
                },
                "parameter": {
                    "type": "string",
                    "description": "The parameter passed as argument to the function.",
                    "example": "Point-01"
                }
            }
        }
    }
}
```


Figure 5. 23 JSON schema for Function type

This schema is stored online inside one of the author's GitHub repositories. Figure 5.24 shows how functions can be defined in a JSON schema of a class. Following the UML2JSON best practice document's Requirement 2 about the way to reference types from the external schema, the external schema can be referred as follow: *https://raw.githubusercontent.com/rifqialf/conceptual-design-gwa/main/uml_classoperation_schema_definition.json#/$def/Function.*

```
"BusStop": {
    "$anchor": "BusStop",
    "type": {…
    },
    "properties": {
        "geometry": {…
        },
        "sourceCRS": {…
        },
        "relevantAttributes": {…
        },
        "attributeFilter": {
            "type": {
                "$ref": "https://raw.githubusercontent.com/rifqialf/conceptual-design-gwa/main/uml_classoperation_schema_definition.json#/$def/Function"
            },
            "minitems": 0,
            "maxitems": 1,
            "default": {
                "attribute": {
                    "type": {
                        "$ref": "#/$defs/CityName"
                    },
                    "default": ""
                }
            }
        },
        "isOfferedBy": {…
        }
    },
    "required": […
    ]
```


Figure 5. 24 Example of PIM function JSON pointer

5.4.1.4. JSON Schema for Association Roles

Requirement #5.4.1.4

Association name shall be assigned onto the related class as association roles by assuming the active and passive voice of the association name. The assumption is performed based on observing the arrowhead of the association name. The active-voiced association role is placed close to the class that is pointed by the arrowhead while the passive-voiced association role resides the opposite class.

Resource: PIM-Profiles in JSON format (provided); Execute: Manual by User

The current UML2JSON best practice document only covers the way association roles can be converted into JSON encoding, but not association names. In an association relationship, association roles are the semantics that specify the role of each class in said relationship. In summary, the current UML2JSON best practice document requires an association to include two distinct words to represent the two involved classes, while each association in the current PIM development only configure one association name. Based on this limitation, an additional step is needed to convert the single-word association names into two-words association roles.

As explained in previous section, the semantics of association stereotypes are a single verb with additional arrow that indicates the way the association name shall be used. This arrow notion helps to generate association roles by converting the single-word association name into both active and passive voice (e.g. display association name is converted into "displays" and "is displayed by"). The rule is that the active voice form of the association name is placed close to the class that is pointed by the arrow.

Figure 5.25 shows the way display association name is converted into two association roles. Figure 5.26 shows example of isOfferedBy, a result of association roles conversion, in JSON schema according to the best practice document.

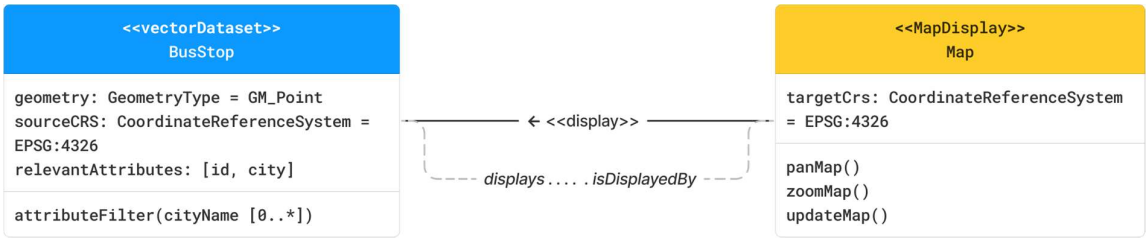


Figure 5. 25 Example of association name being converted to association roles.

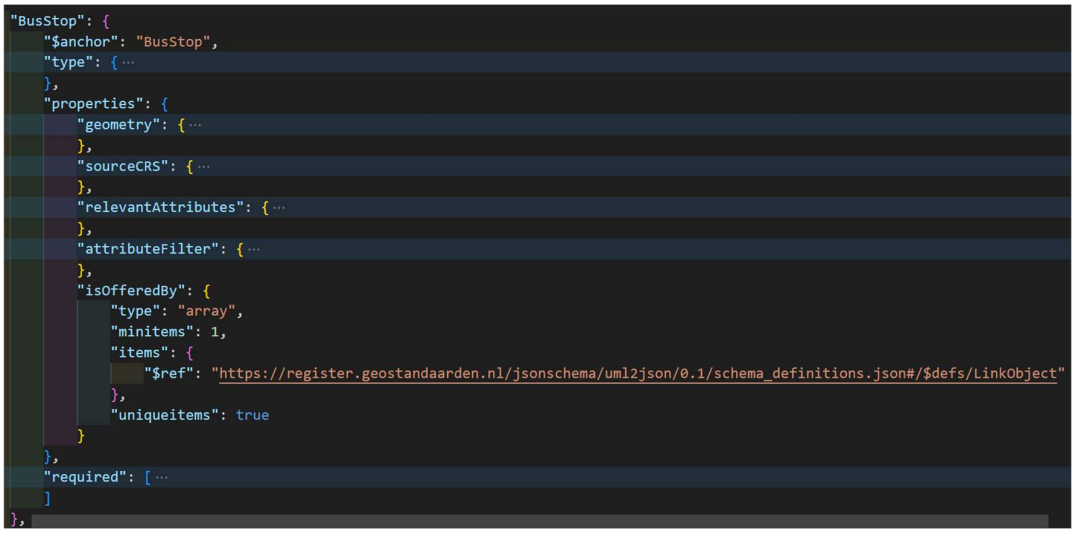


Figure 5. 26 Example of association role stored in JSON format.

After all elements in the PIM diagram have been converted into JSON format, the transformation continues in the integrated development environment of your choice. This exact step will be elaborated later in MDA Implementation section. The following sections elaborates on how to map the elements of PIM into PSM.

5.4.2. Transformation Mapping

This section explains about how the JSON-converted PIM transforms into PSM. The elements of PIM stereotypes were mapped into elements of PSM stereotypes for showing the logic behind each element transformation. The following subsections explains several considerations that were elaborated in developing the transformation such as the inheritance, default values, and implications on specific decisions. Figure 5.27 shows the parts of research flowchart that are relevant to this section.

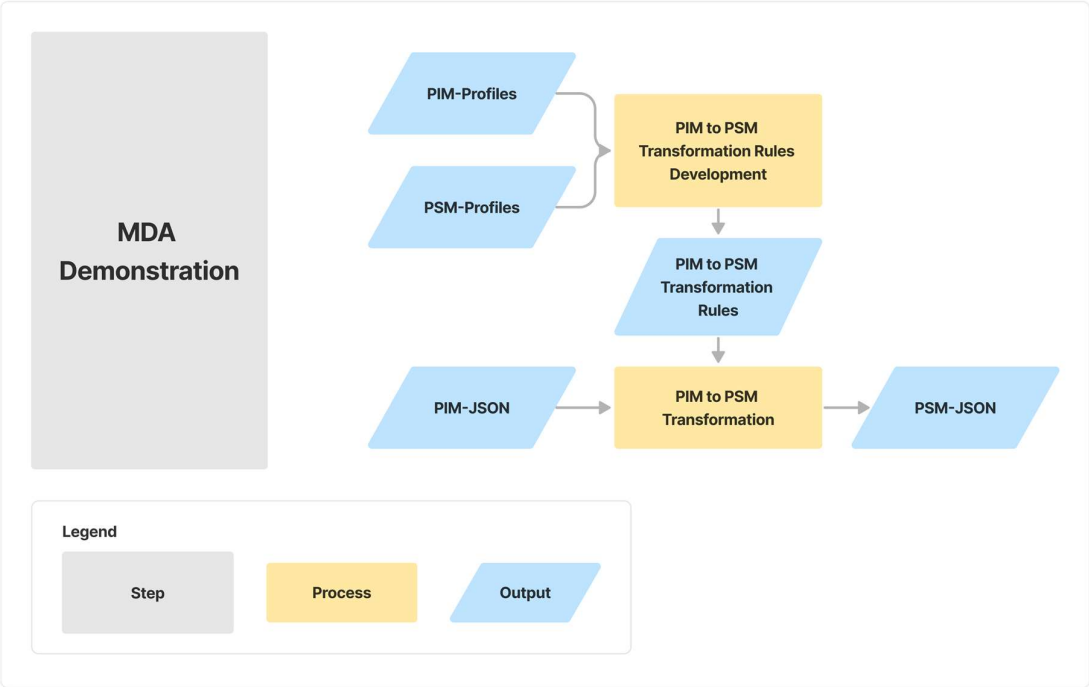


Figure 5. 27 Part of Research Flowchart that are related to PIM transformation mapping.

Several requirements were developed to help users complete the PIM transformation mapping according to the proposed approach. Following the requirements below in sequence allows users to map PIM elements into PSM elements. Figure 5.28 shows the overview of the requirements used for PIM transformation mapping.

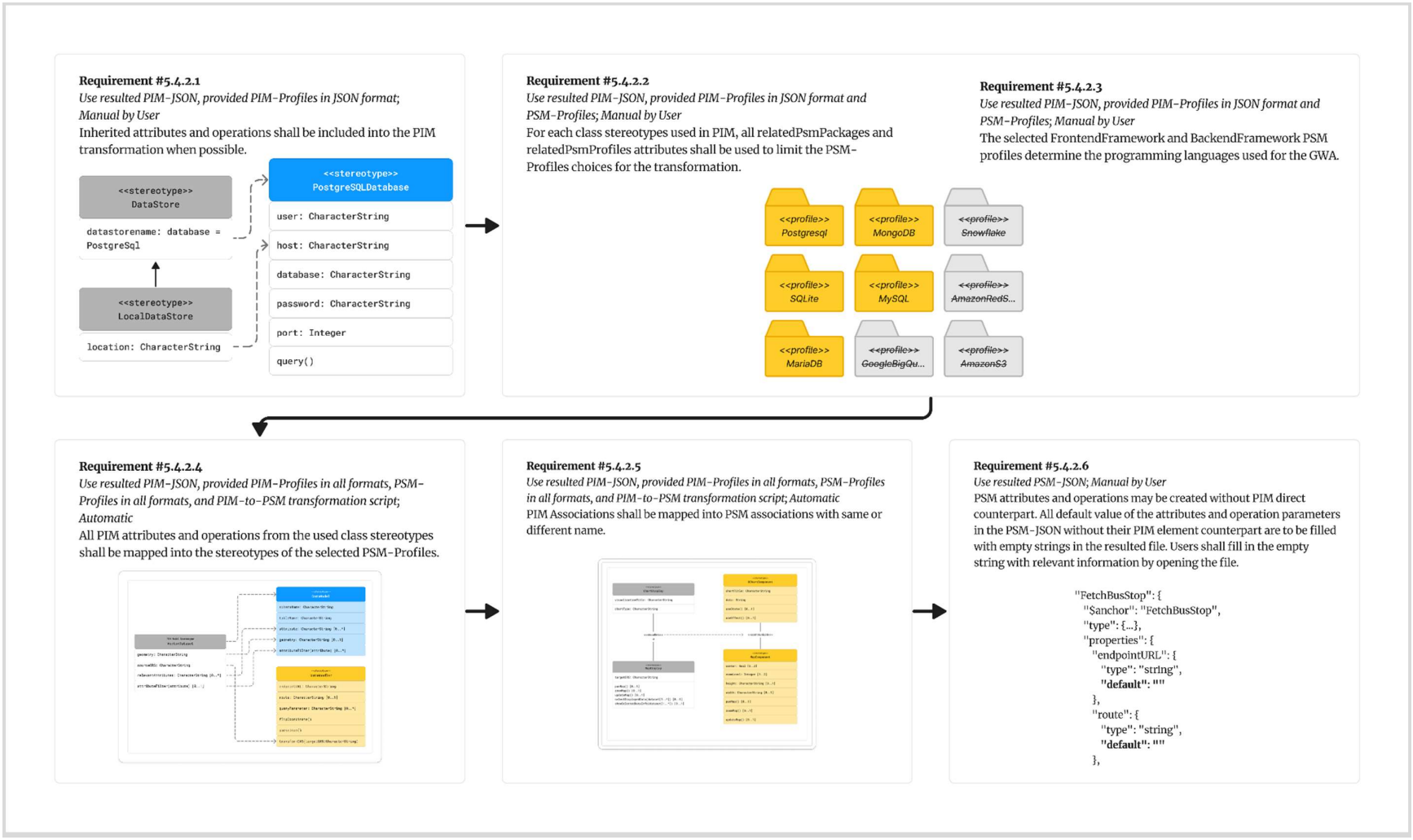


Figure 5. 28 Overview of requirements for PIM transformation mapping

5.4.2.1. Inherited Attributes and Operations

Requirement #5.4.2.1
Inherited attributes and operations shall be included into the PIM transformation when possible.

Resource: PIM-JSON (from JSON conversion), PIM-Profiles in JSON format (provided); Execute: Manual by User

The first consideration on the PIM transformation is to ensure all PIM classes to include inherited attributes and operations from the relevant parent classes, except for optional attributes and operations. Figure 5.29 shows an example of how *LocalDataStore* class stereotype inherits *DataStore* class stereotype's attribute and is included the mapping to PSM elements.

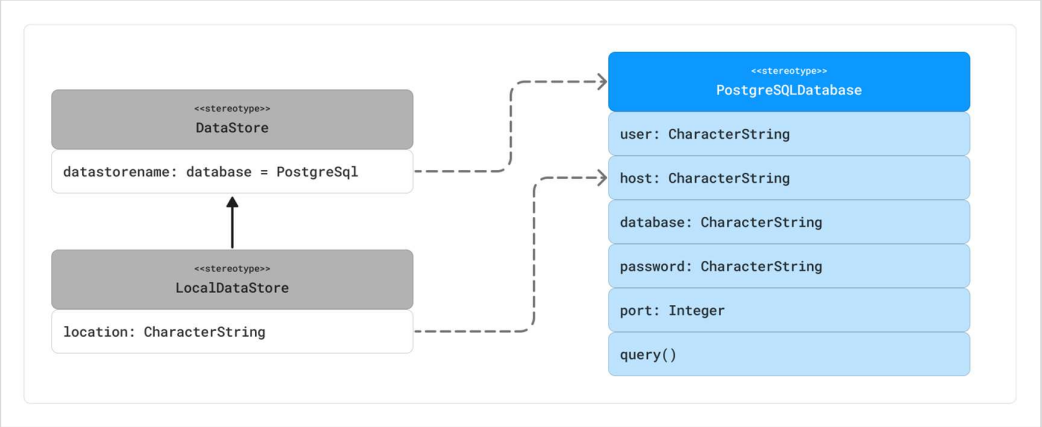


Figure 5. 29 Example of inherited attributes and operations in PIM class

5.4.2.2. PSM Profile Choices for Transformation

Requirement #5.4.2.2
For each class stereotypes used in PIM, all relatedPsmPackages and relatedPsmProfiles attributes shall be used to limit the PSM-Profiles choices for the transformation.

Resource: PIM-JSON (from JSON conversion), PIM-Profiles in JSON format (provided), PSM-Profiles (provided); Execute: Manual by User

The previous Requirement #5.4.2.1 also includes the inheritance of the private attributes *relatedPsmPackages* and *relatedPsmProfiles*. These two attributes contribute to the PIM transformation by providing the information of which PSM-Profiles are available to transform to. This approach gives limitation for user to not choosing the technologies that is not appropriate to the used stereotype in PIM. The remaining PSM-Profiles Figure 5.30 shows an illustration on how the *LocalDataStore* class stereotype's *relatedPsmProfiles* eliminates several PSM-Profiles to be used.

In the context of choosing the PSM-Profiles for the transformation, another consideration is the dependency of the technologies to each other, which is determined to be outside of the scope of this research. Even though any technology combinations are possible to be selected, selecting technologies without considering their compatibility and language disparity to each other might increase the chance of

encountering dependency-related problems. In this research, this consideration is handed to the user to choose wisely.

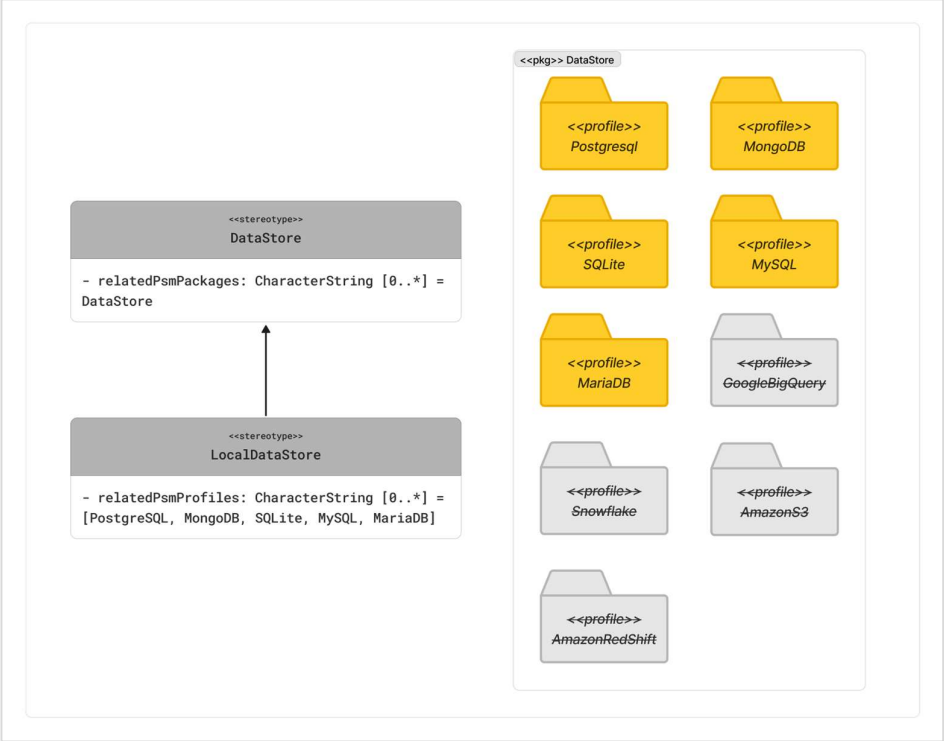


Figure 5. 30 Example of curating the PSM profiles using the private attribute.

5.4.2.3. PSM Programming Language Choice

Requirement #5.4.2.3
The selected FrontendFramework and BackendFramework PSM profiles determine the programming languages used for the GWA.

Resource: PIM-JSON (from JSON conversion), PIM-Profiles in JSON format (provided), PSM-Profiles (provided);
Execute: Manual by User

The Requirement #5.4.2.2 also allows users to select the PSM profiles from *FrontendFramework* and *BackendFramework* packages. In this research, the decision on these packages also effectively determines the programming language associated with each framework. For example, choosing React, a frontend framework primarily used with JavaScript, means that the developed GWA will use JavaScript as the programming language of choice. Another example is that choosing Laravel, a PHP-based backend framework, means that the developed GWA will use PHP as the programming language.

It is important to remind that when users chose the combination of frameworks, the potential dependency-related problems mentioned in Requirement #5.4.2.2 might still arise. There is no preferred

sequence for the order on choosing the *FrontendFramework* or *BackendFramework* profile first. Users are suggested to consider the framework dependency wisely.

5.4.2.4. PIM Attribute and Operation Mapping

Requirement #5.4.2.4

All PIM attributes and operations from the used class stereotypes shall be mapped into the stereotypes of the selected PSM-Profiles.

Resource: PIM-JSON (from JSON conversion), PIM-Profiles in all formats (provided), PSM-Profiles in all formats (provided); PIM-to-PSM transformation script (provided); Execute: Automatic

The next consideration is to ensure that all attributes and operations defined within the PIM must be mapped to corresponding elements in the PSM. This requirement ensures that the model transformations do not allow any loss of information. The attributes and operations from a single PIM class may be mapped to separate instance of PSM classes. Figure 5.31 shows an example of a single VectorDataset class stereotype which attributes and operations contribute to two separate PSM class stereotypes in the transformation.

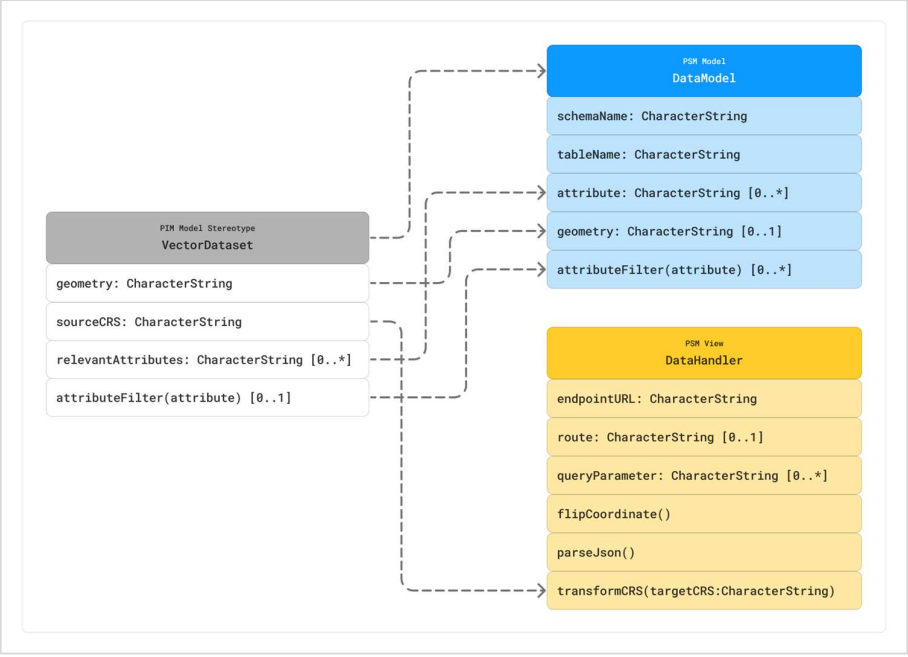


Figure 5. 31 Example of PIM attribute mapping

5.4.2.5. PIM Association Mapping

Requirement #5.4.2.5

PIM Associations shall be mapped into PSM associations with same or different name.

Resource: PIM-JSON (from JSON conversion), PIM-Profiles in all formats (provided), PSM-Profiles in all formats (provided); PIM-to-PSM transformation script (provided); Execute: Automatic

Alongside of PIM classes, associations are also required to be mapped within the PSM. While the specific semantics might differ between PIM and PSM, the core purpose of which MVC layers are being associated with should be preserved. Figure 5.32 shows an example of straightforward association transformation of visuallink which associate chart component and map component in this case.

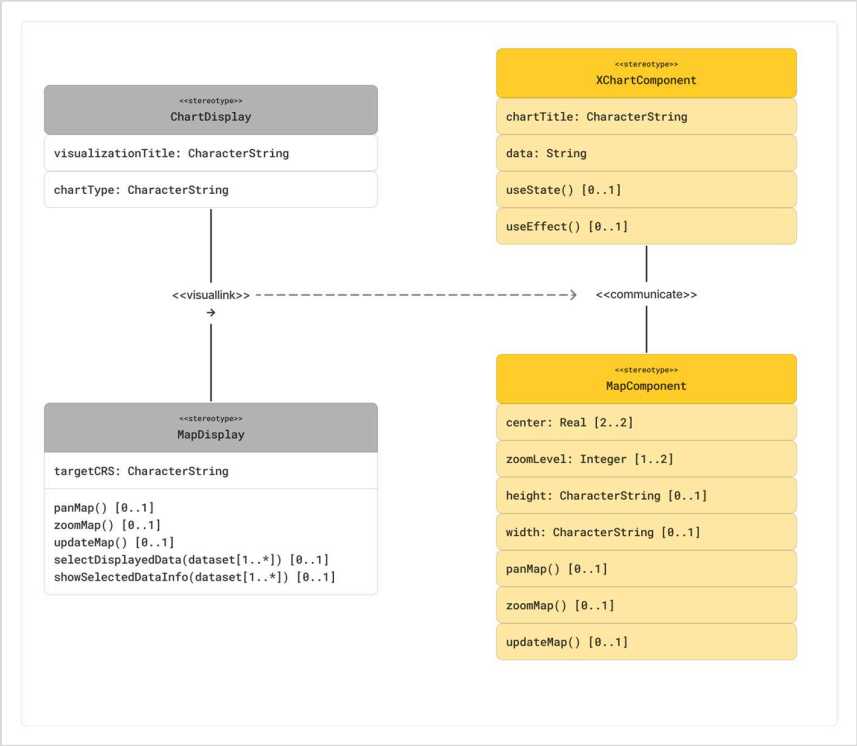


Figure 5. 32 Example of association mapping

5.4.2.6. PSM Classes' Default Values

Requirement #5.4.2.6

PSM attributes and operations may be created without PIM direct counterpart. All default value of the attributes and operation parameters in the PSM-JSON without their PIM element counterpart are to be filled with empty strings in the resulted file. Users shall fill in the empty string with relevant information by opening the file.

Resource: PSM-JSON (result of transformation); Execute: Manual by User

The leading condition to the PIM element mapping to PSM is that the Platform Specific Model (PSM) may not directly mirror the structure of the Platform Independent Model (PIM) with one-to-one class and element correspondences. This means that during the transformation of PSM class, some of the PSM classes' attributes and operations may not have default value in the PSM-JSON.

The empty default values become relevant in the end of the PIM transformation. After the resulted PSM-JSON file is generated from the transformation, users shall check the file and fill in all empty default values. Although keeping the default value empty will not cause the PSM-to-code transformation to fail, empty values might cause some parts of the transformation scripts to not run properly or return incomplete code. Figure 5.33 shows how PIM class transformations may leave PSM attributes and operations untouched, while Figure 5.34 shows an example of several default values in PSM-JSON of PostgreSQL class are filled with empty string after the transformation.

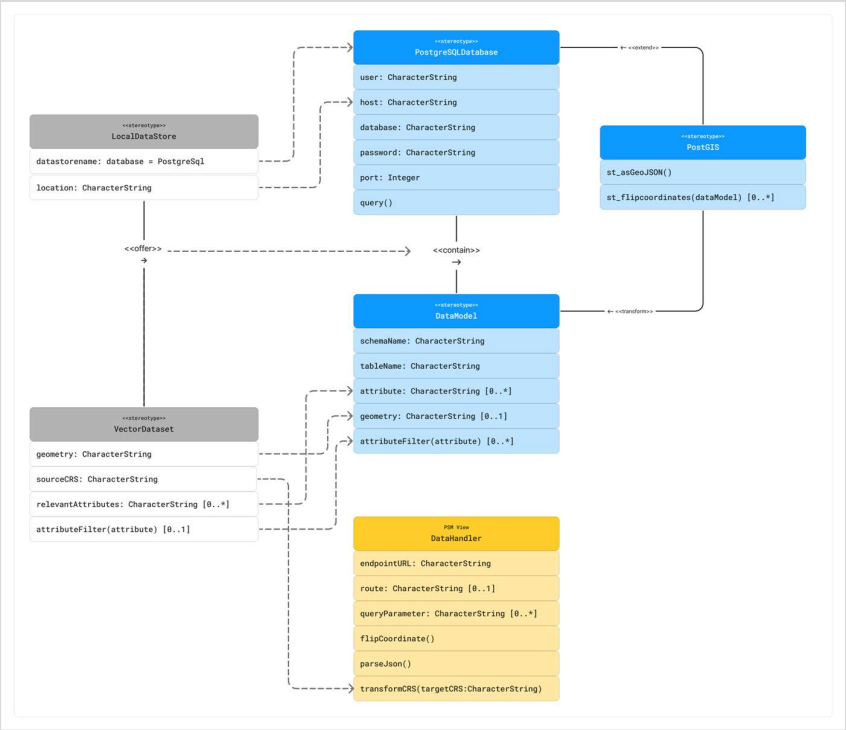


Figure 5. 33 Example of PSM elements being present without PIM counterpart.

Figure 5.33 above also showcases another interpretation of PSM elements that have no PIM counterpart in the transformation. In that figure, the PostGIS class are present in the transformation despite having no relationship with any PIM elements. This type of class and association instantiations are possible depending on the specification of selected platforms and technologies.

Figure 5. 34 Example of empty string in PSM class

For example, the PostGIS class instantiation in the case portraited in the Figure 5.34 is plausible as PostGIS is an extension of PostgreSQL which is derived from PIM elements. The PostGIS class is generated without any condition during the PIM transformation except on the sole fact that PostgreSQL required PostGIS to handle geospatial data. The interpretations of the Requirement 008 like explained above require the user's understanding of the selected platforms and technologies.

5.4.3. Model Transformation Script

This section explains about how the transformation takes place using the prepared transformation script developed using JavaScript code in node.js environment. After the users convert their designed PIM into JSON schema, they are directed to integrate their PIM-JSON file into the transformation scripts to perform the transformations. The following subsections explains what the transformation scripts do in the transformation. This section covered in the Figure 5.35 in the research flowchart, where the figure shows the parts of research flowchart that are relevant to this section.

Several requirements were developed to help users complete the PIM-to-PSM model transformation script development according to the proposed approach. Following the requirements below in sequence allows users to create a PIM-to-PSM model transformation script development that can execute the model transformation and generate PSM. Figure 5.35 shows the overview of the requirements used for PIM instantiation.

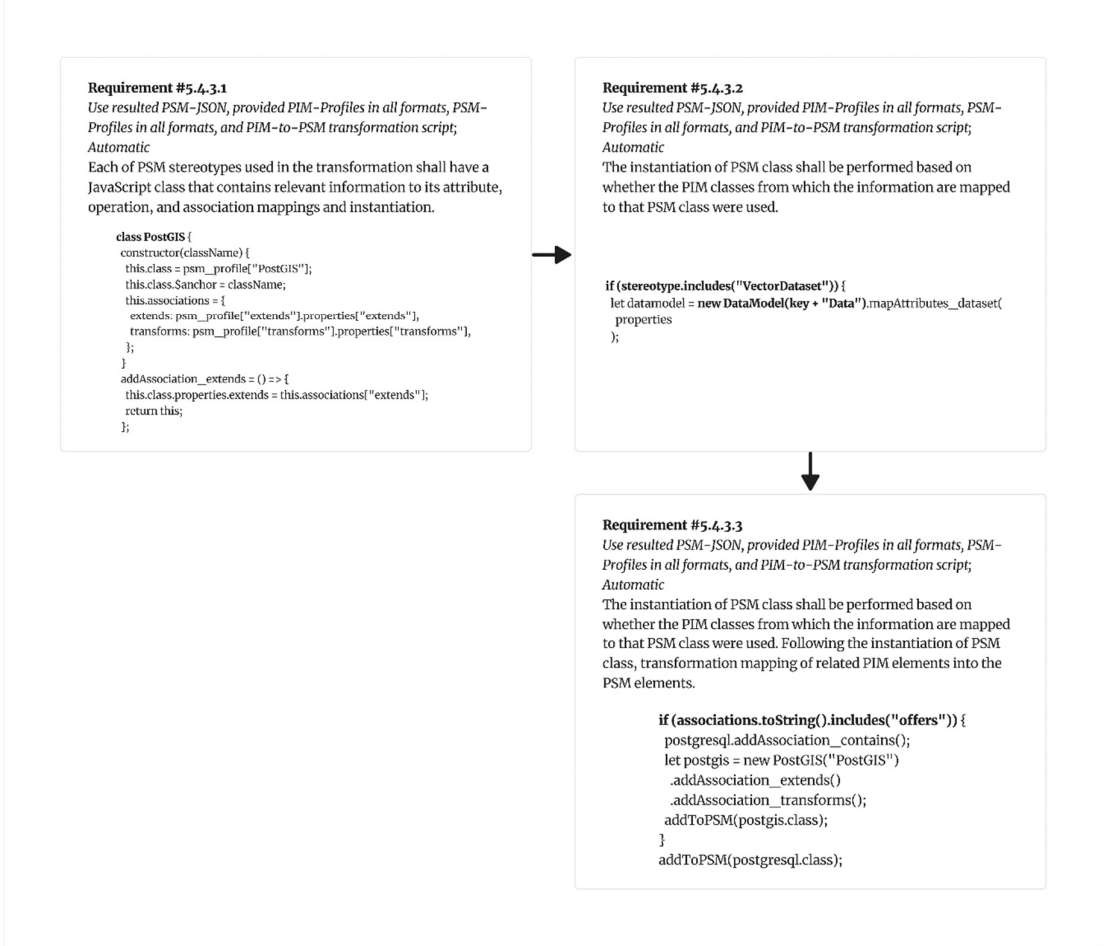


Figure 5. 35 Overview of requirements for Model Transformation Script Development

5.4.3.1. PSM Stereotype Transformation as JavaScript Classes

Requirement #5.4.3.1

Each of PSM stereotypes used in the transformation shall have a JavaScript class that contains relevant information to its attribute, operation, and association mappings and instantiation.

Resource: PSM-JSON (from transformation), PIM-Profiles in all formats (provided), PSM-Profiles in all formats (provided); PIM-to-PSM transformation script (provided); Execute: Automatic

The transformation scripts handle all possible PIM stereotype transformations into the PSM stereotypes contained in the selected profiles. JavaScript classes were used to handle the mapping and instantiation of each PSM stereotypes. The JavaScript classes contain information such as the JSON schema of the PSM stereotype and possible associations for the PSM stereotype. The JavaScript classes also handle the logic of how the PSM stereotype attributes, operations, and associations are mapped based on information come from the relevant PIM. Figure 5.28 shows the JavaScript class of the PostgreSQLDatabase PSM stereotype as example.

```
class PostgreSQLDatabase {
  constructor(className) {
    this.class = psm_profile["PostgreSQLDatabase"];
    this.class.$anchor = className;
    this.associations = {
      isExtendedBy: psm_profile["isExtendedBy"].properties["isExtendedBy"],
      contains: psm_profile["contains"].properties["contains"],
    };
  }

  addAssociation_isExtendedBy = () => {···
  };

  addAssociation_contains = () => {···
  };

  mapAttributes_datastore = (class_datastore) => {···
  };
}
```

Figure 5. 36 Example of JavaScript class of PSM stereotype

5.4.3.2. PSM Class Instantiation from Stereotype

Requirement #5.4.3.2

The instantiation of PSM class shall be performed based on whether the PIM classes from which the information are mapped to that PSM class were used.

Resource: PSM-JSON (from transformation), PIM-Profiles in all formats (provided), PSM-Profiles in all formats (provided); PIM-to-PSM transformation script (provided); Execute: Automatic

Based on the developed JavaScript class, the PSM class were instantiated. The instantiation is based on whether the PIM stereotypes from which information are mapped to the PSM class exist in the user-provided PIM-JSON file. Figure 5.37 shows the example of the instantiation of PSM classes of DataModel and DataHandler stereotypes based on whether PIM VectorDataset exists in the PIM-JSON file.

```
if (stereotype.includes("VectorDataset")) {
  let datamodel = new DataModel(key + "Data").mapAttributes_dataset(
    properties
  );
  let datahandler = new DataHandler("Fetch" + key).mapAttributes_dataset(
    properties
  );
```

Figure 5. 37 Example of how PSM classes are instantiated based on PIM stereotype.

5.4.3.3. Conditional Mapping

Requirement #5.4.3.3

The instantiation of PSM class shall be performed based on whether the PIM classes from which the information are mapped to that PSM class were used. Following the instantiation of PSM class, transformation mapping of related PIM elements into the PSM elements.

Resource: PSM-JSON (from transformation), PIM-Profiles in all formats (provided), PSM-Profiles in all formats (provided); PIM-to-PSM transformation script (provided); Execute: Automatic

In addition to the instantiation of the PSM class, the information mapping may also be performed between the used PIM stereotypes in PIM-JSON file and the instantiated PSM class based on other mapped conditions. As previously elaborated in the Transformation Mapping section, each information contained in the PIM class can contribute to the instantiations of attributes, operations, and associations of the relevant PSM classes, as well as to the instantiations of a new PSM class. This requirement commonly used to transform mapped associations between the PIM class and the related PSM class, although it may not be only limited to association mapping. Figure 5.38 shows an example of this condition where the offer association from PIM-JSON may transformed into the contains association in the relevant PostgreSQLDatabase class as well as instantiate a new PostGIS class along with its associations.

```
if (associations.toString().includes("offers")) {
  postgresql.addAssociation_contains();

  let postgis = new PostGIS("PostGIS")
    .addAssociation_extends()
    .addAssociation_transforms();
  addToPSM(postgis.class);
}
addToPSM(postgresql.class);
```

Figure 5. 38 Example of mapping conditions

5.5. PSM to Code Transformation Rules

This section explains about how to create a model transformation of PSM into the computer code of GWA. This step was performed with reproducibility in mind. The most suitable target users to reproduce this step are User Type #2: The Developers. By knowing how to perform the code generation, users can reproduce the step to transform their PSM into the computer code of their selected technology. The rest of this section elaborates on all steps and considerations in developing the model transformation.

5.5.1. Transformation Mapping

This section explains about how the transformed PSM-JSON transforms into the computer code of determined programming language. This is the second model transformation in the MDA implementation in this research. After users performed PIM-to-PSM transformation and filled in the default values in the resulted PSM-JSON file, the PSM-JSON file is brought to the PSM-to-code transformation script as input. The elements of PSM stereotypes were mapped into the relevant computer code. Figure 5.39 shows the parts of research flowchart that are relevant to this section.

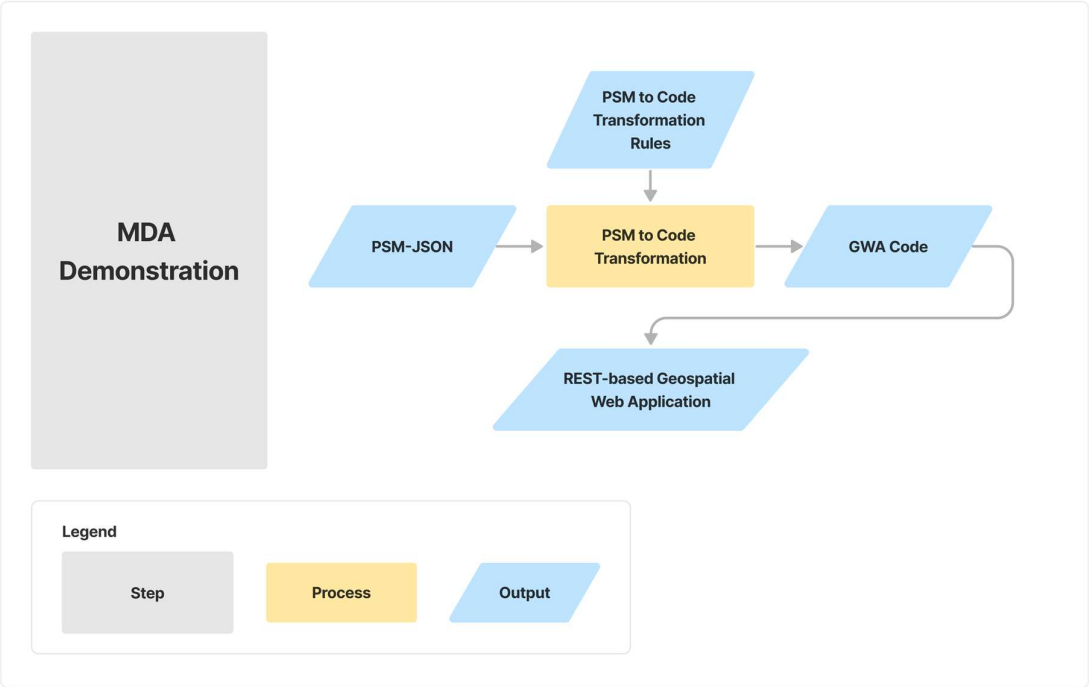


Figure 5. 39 Part of Research Flowchart that are related to PIM Instantiation

Several requirements were developed to help users complete the PSM transformation mapping according to the proposed approach. Following the requirements below in sequence allows users to map PSM elements into source code file. Figure 5.40 shows the overview of the requirements used for PSM transformation mapping.

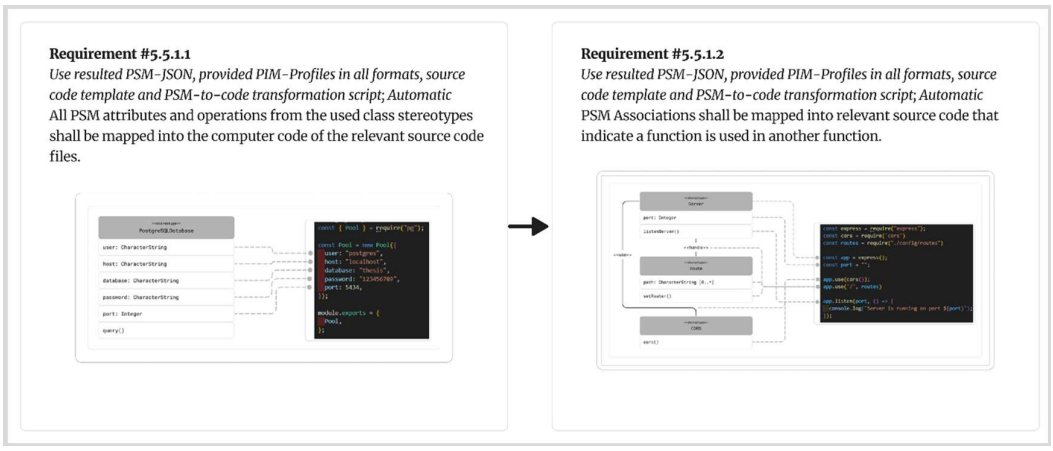


Figure 5. 40 Overview of requirements for PSM transformation mapping

For the target of the transformation, the directory structure to store the source code files were proposed. The principle of "convention over configuration" were applied in the proposal, meaning that the decided directory structure and many of the terminologies used within it follow the common conventions in software development processes. This principle helped in simplify the transformation process by deciding the conventional directory structure instead of coming up with another explicit terminologies. The examples of those conventions are as follows:

1. The folder names 'model', 'view', and 'controller' are common folder names used in MVC-based projects.
2. The file names 'index.html' for default entry point for web application and 'server.js' for main server file in Node.js are commonly used terminology in web development project.

Figure 5.41 shows the proposed directory and the structures for the transformation mapping. The following subsections explains the considerations in this model transformation step.

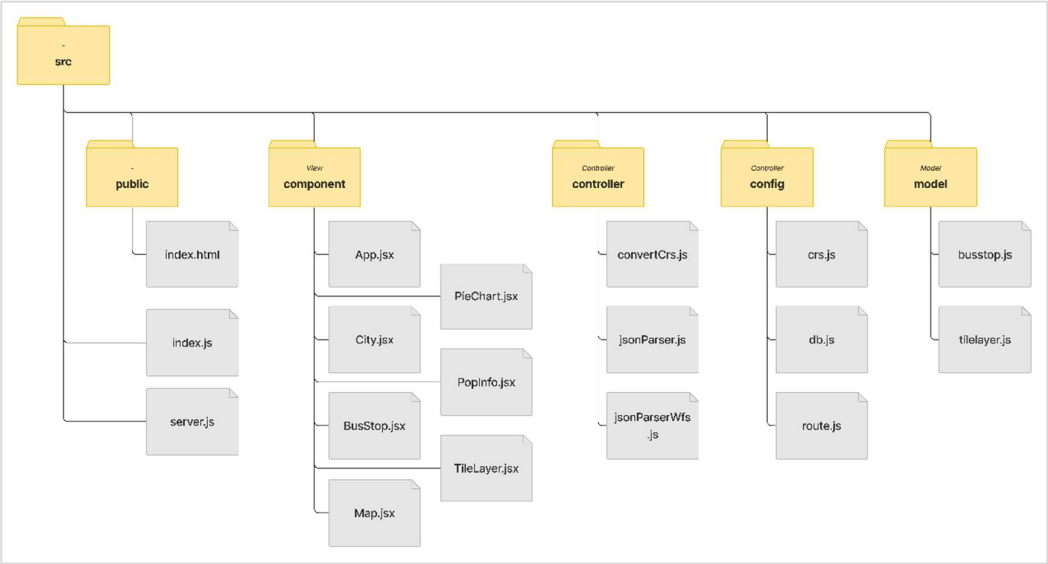


Figure 5. 41 The proposed directory structure of the GWA source code files

5.5.1.1. PSM Attribute and Operation Mapping

Requirement #5.5.1.1

All PSM attributes and operations from the used class stereotypes shall be mapped into the computer code of the relevant source code files.

Resource: PSM-JSON (from transformation), PSM-Profiles in all formats (provided), Source code template (provided); PSM-to-code transformation script (provided); Execute: Automatic

Like the previously explained PIM-to-PSM transformation, in this PSM-to-code transformation, the elements of PSM class stereotypes were mapped into computer code based on its relevant source code files. Based on the transformation mapping process during the previous PIM-to-PSM transformation, PSM classes in the resulted PSM-JSON file may already came with inherited attributes and operations

from their parent classes. This situation makes transformation mapping in this step easier as there is less responsibility for the users to check whether inherited properties were used properly. Nevertheless, user discretion is still suggested.

Figure 5.42 shows the example of PostgreSQLDatabase PSM class stereotype where its attributes and operations were mapped into an instance of db.js containing source code for creating a pool connection to PostgreSQL.

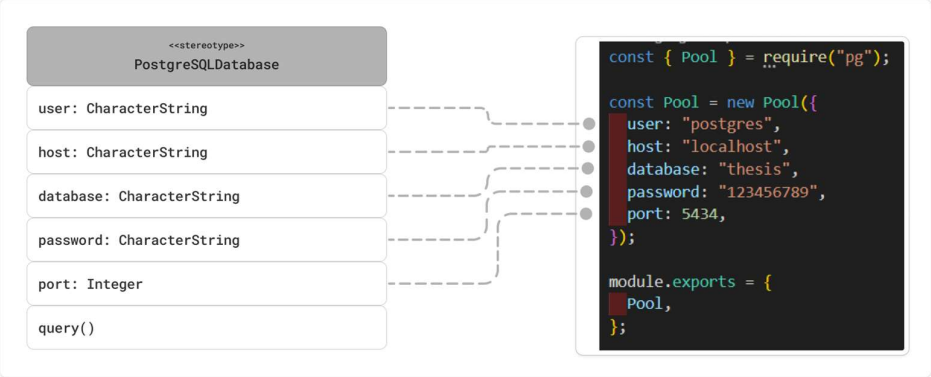


Figure 5. 42 Example of PSM class being mapped into computer code

5.5.1.2. PSM Association Mapping

Requirement #5.5.1.2
PSM Associations shall be mapped into relevant source code that indicate a function is used in another function.

Resource: PSM-JSON (from transformation), PSM-Profiles in all formats (provided), Source code template (provided); PSM-to-code transformation script (provided); Execute: Automatic

Like the previously explained PIM-to-PSM transformation, in this PSM-to-code transformation, the elements of PSM association stereotypes were mapped into computer code based on its relevant source code files. The association stereotypes used in the PSM-JSON shall be mapped into a part of the source code that indicate that the PSM classes that use the association are related to each other in a certain way. This relationship might be that a function is a part of another function, or the result of a function is used in another function. The mapping condition is determined by the user during the development of the transformation rules. Figure 5.43 shows the example of server.js source code generation that is the result of association mapping between several PSM classes.

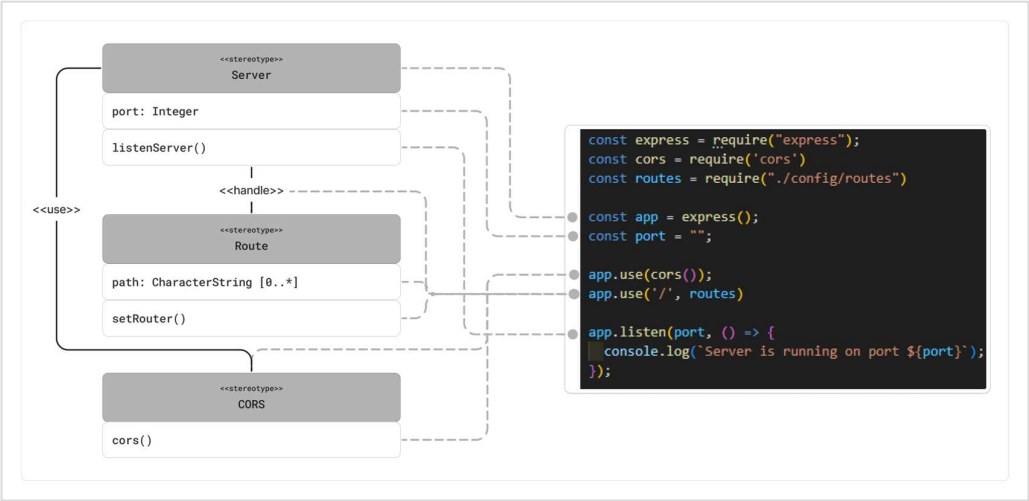


Figure 5. 43 Example of PSM association mapping into computer code

5.5.2. Model Transformation Script

This section explains about how the PSM-to-code transformation is to be performed using JavaScript code executed in node.js environment. After the users performed PIM-to-PSM transformation and filled in the empty default values in the resulted PSM-JSON file, the users are directed into the PSM-to-code transformation script. The following subsections explains what the transformation scripts do in the transformation. This section covered in the Figure 5.39 in the research flowchart, where the figure shows the parts of research flowchart that are relevant to this section.

Several requirements were developed to help users complete the PSM-to-code model transformation script development according to the proposed approach. Following the requirements below in sequence allows users to create a PSM-to-code model transformation script development that can execute the model transformation and generate the source code of the GWA. Figure 5.44 shows the overview of the requirements used for PIM instantiation.

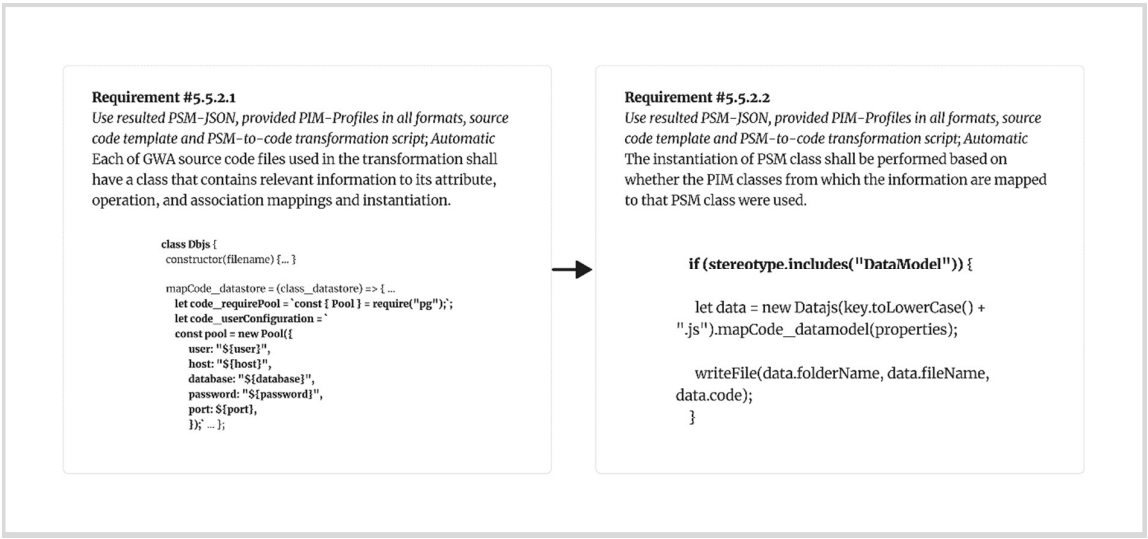


Figure 5. 44 Overview of requirements for PSM-to-code model transformation script development.

5.5.2.1. JavaScript File-based Classes

Requirement #5.5.2.1

Each of GWA source code files used in the transformation shall have a class that contains relevant information to its attribute, operation, and association mappings and instantiation.

Resource: PSM-JSON (from transformation), PSM-Profiles in all formats (provided), Source code template (provided); PSM-to-code transformation script (provided); Execute: Automatic

The PSM-to-code transformation scripts handle all possible PSM stereotype transformations into the computer code of the selected technologies and programming language. Similar to the PIM-to-PSM transformation step, JavaScript classes were used to handle the mapping and instantiation of each source code files and their contents. Figure 5.45 shows the example of *Datajs* JavaScript class that handles how data-related source code files are created.

```
class Datajs {
  constructor(filename) {
    this.folderName = "gwa/model/";
    this.fileName = filename;
    this.code = "";
  }
  mapCode_datamodel = (class_datamodel) => {
    let sourceToCode = "";
    let attributesToCode = "";
    let attributeFilterToCode = "";

    const add_schemaName_tableName = (class_datamodel) => {···
    };

    const add_propertyNames = (class_datamodel) => {···
    };

    const add_attributeFilter = (class_datamodel) => {···
    };

    const create_script = (class_datamodel) => {···
    };

    // Execution
    add_schemaName_tableName(class_datamodel);
    add_propertyNames(class_datamodel);
    add_attributeFilter(class_datamodel);
    create_script(class_datamodel);

    return this;
  };
}
```

Figure 5. 45 Example of JavaScript class to handle source code mapping from PSM classes.

5.5.2.2. Code Template Generation

Requirement #5.5.2.2

The instantiation of PSM class shall be performed based on whether the PIM classes from which the information are mapped to that PSM class were used.

Resource: PSM-JSON (from transformation), PSM-Profiles in all formats (provided), Source code template (provided); PSM-to-code transformation script (provided); Execute: Automatic

Within the JavaScript classes, the template of the script of each file were provided. The code templates come with designated template literals which fill in information that come from the mapped PSM-JSON information. Separation of the code into several variables might be performed in case parts of the code should be instantiated in different conditions. Figure 5.46 shows the example of *Dbjs* JavaScript class' *mapCode_datastore* method that contains the script for a db.js file for connecting the GWA with PostgreSQL database via pool connection.

This code template generation requirement also covers the conditional mapping in this model transformation. Figure 5.47 shows the example of how the Dbjs JavaScript class is instantiated depending on whether the PostgreSQLDatabase PSM class exists in the provided PSM-JSON file.

```
class Dbjs {
  mapCode_datastore = (class_datastore) => {
    let user = class_datastore.user.default;
    let host = class_datastore.host.default;
    let database = class_datastore.database.default;
    let password = class_datastore.password.default;
    let port = class_datastore.port.default;

    const create_script = () => {
      let code_requirePool = `const { Pool } = require("pg");`;
      let code_userConfiguration = `
      const pool = new Pool({
            user: "${user}",
            host: "${host}",
            database: "${database}",
            password: "${password}",
            port: ${port},
            });`;
      let code_exportModule = `module.exports = { pool };`;

      let code = [code_requirePool, code_userConfiguration, code_exportModule];

      code = code.join("\n");
      this.code = code;
    };
```

Figure 5. 46 Example of code template

```
if (stereotype.includes("PostgreSQLDatabase")) {
  let pool = new Dbjs("db.js").mapCode_datastore(properties);
  writeFile(pool.folderName, pool.fileName, pool.code);
}
```

Figure 5. 47 Example of how source code is instantiated into source code file based on PSM information

6. MDA IMPLEMENTATION

This chapter uses the developed UML profiles and model transformation rules from previous chapter into an MDA implementation. The implementation aims to showcase the value of the developed approach in REST-based GWA development. First, a GWA example was introduced as the aimed result of the implementation. Then, the implementation used the developed UML profiles to instantiate a PIM which then brought into model transformations until the computer code of the GWA was generated. All steps in this chapter were taken and documented with their reproducibility in mind. The appropriate user types were mentioned at the start of each section.

A REST-based GWA example was developed to be the aim for the MDA implementation. Figure 6.1 shows the screenshot of the GWA.

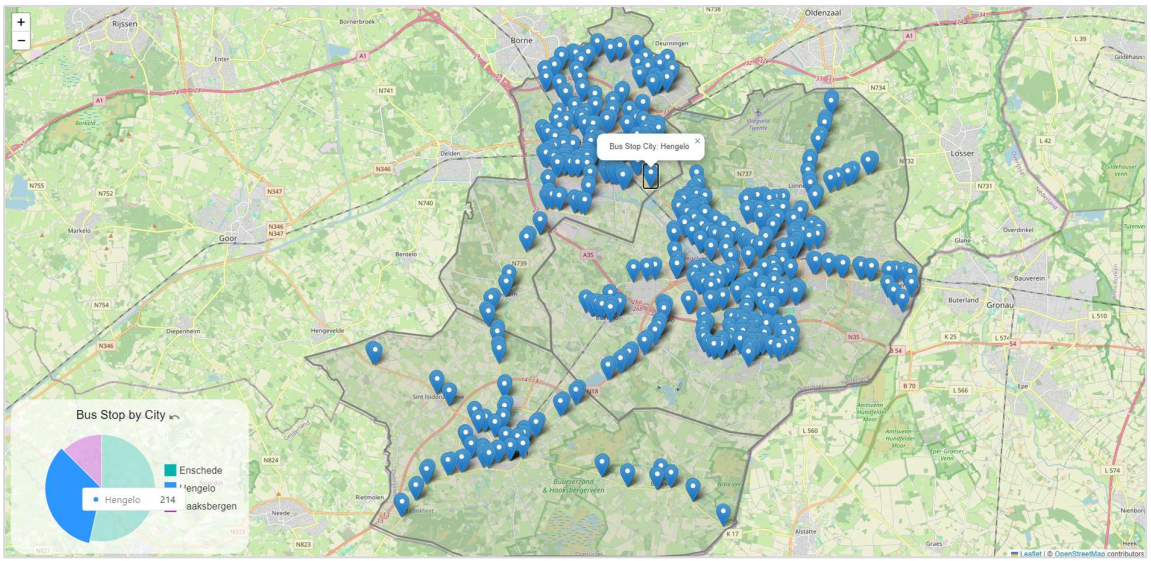


Figure 6. 1 Mock-up of the GWA example

Below are the functionalities that the GWA provides:

1. The GWA shows bus stops within the Dutch city of Hengelo, Enschede, and Haaksbergen as points. The data is downloaded from datashop24.com (Datashop24.com, n.d.) and is provided locally on PostgreSQL database.
2. The GWA shows the three cities' boundaries as polygons. The data is provided as WFS from the open-access PDOK data portal (*Home - PDOK*, n.d.)
3. The GWA uses OpenStreetMap map tile layer as its only basemap.
4. The bus stop points show a pop-up text of the city name the point falls into when clicked.
5. The city boundary polygons show a pop-up text of the city name it represents when clicked.
6. The view of the GWA shows all the data during initialization and can be zoomed and panned by the users.
7. The chart component represents the total number of bus stops in the GWA.
8. The chart shows the total number of bus stops of each city when the mouse hovers each pie of the chart.
9. When a pie of the chart is clicked, the GWA filters the bus stop and city boundary data on the map to only show the data of and within the selected city.
10. The chart provides a button to reset the city selection to reset the GWA to show all data.

Below are the technical requirements of the developed GWA example:

1. It uses React-Leaflet as frontend framework. It is a wrapper that binds React frontend framework with Leaflet as interactive map visualization library. This technology is chosen for the GWA example because React is the most popular frontend framework according to trend survey by Stack Overflow, the popular Q&A forum for developers (Stack Overflow Trends, 2023); and Leaflet is the leading contender of open-source JavaScript library for interactive maps.
2. It uses Material UI for creating the pie chart component. It is a free React UI component provider that comprise of many types of UI components, including charts. This technology is chosen for the GWA example because during the research was conducted, Material UI is one of the most popular UI component libraries with 1.3 million weekly downloads according to the NPM website, the package manager for Node.js.
3. It uses Express as backend framework in Node.js. Express is a Node.js based web application framework that is popular due to its lightweight size and flexibility for developing web applications and mobile applications in Node.js. This technology is chosen for the GWA example because it has been a popular choice for Node.js developers with its full capabilities of component modification.
4. It uses Axios as HTTP client library. This technology is chosen for the GWA example because it has been a popular choice amongst developers for its simplified way to integrate HTTP protocol into applications.
5. It uses PostgreSQL database along with PostGIS extension for handling geospatial data, where the bus stop data of the entire Netherlands is stored. PostgreSQL is chosen for the GWA example because it is the most popular choice for database according to Stack Overflow Developer Survey in 2023 (Stack Overflow, 2023), indicating it as a very relevant technology for implementation.

REST architectural style was adhered by the GWA example during its development. Several points that indicate this compliance are as follows:

1. The GWA uses unique URLs for identifying resources. For example, the local bus stop data from PostgreSQL was designed in a way so that it is accessed through */busstop* route.
2. The GWA uses proper HTTP protocol that allows stateless principle, meaning that each request from the client contains all information the server needs to fulfil that request without storing any state information on the server.
3. The GWA uses JSON as the only resource representations for data exchange between client and server.

OGC API Standards and the overlaying OpenAPI Specifications were also adhered by the GWA example during its development. Several points that indicate these compliances are as follows:

1. The GWA uses proper REST architectural style principles, including standardized HTTP protocol and the use of unique URLs, as explained in previous paragraph.
2. The GWA maintains geospatial data models by several ways including using GEOJSON for representing features in the GWA as well as ensuring all geospatial data conforms to one target CRS.

Using the GWA example as the aim, the MDA implementation was performed. The following sections explain how the implementation used the developed UML profiles and followed the developed approach.

6.1. PIM Instantiation

This section demonstrate how PIM instantiation step is performed using the developed transformation rules from previous chapter. This step was performed with reproducibility in mind. The most suitable target users to perform this step are:

- User Type #1: The Designers.
- User Type #2: The Developers.

Figure 6.2 shows the designed PIM for developing the GWA example using stereotypes contained in the developed PIM-Profiles. The requirements were adhered during this PIM design step, elaborated as follows:

Requirement #5.3.1

Each class in the designed PIM used the most specific child stereotypes of each PIM-Profiles, according to the. For example, LocalDataStore class stereotype was used to instantiate BusStopDataStore instead of its more generic parent class stereotype DataStore.

Requirement #5.3.2

The PIM class's naming conventions were also followed.

Requirement #5.3.3

When a stereotype was used, all of its default values were filled in according to its specified attribute type. For example, the *BusStopDataStore* class in the PIM has default values filled in for *datastorename* and *ipAddress* attributes.

Requirement #5.3.4.1 and #5.3.4.2

Several enumerations were created for filling the attribute type of a PIM class. For example, the *BaseMap* class uses *BaseMapProvider* enumeration's string-typed literal on its *provider* attribute.

Requirement #5.3.5

GeometryType enumeration was used in the designed PIM as attribute type of CityBoundary and BusStop classes' geometry attributes.

Requirement #5.3.6 and #5.3.7

All associations used solid lines with association names indicating the reading direction of the relationship.

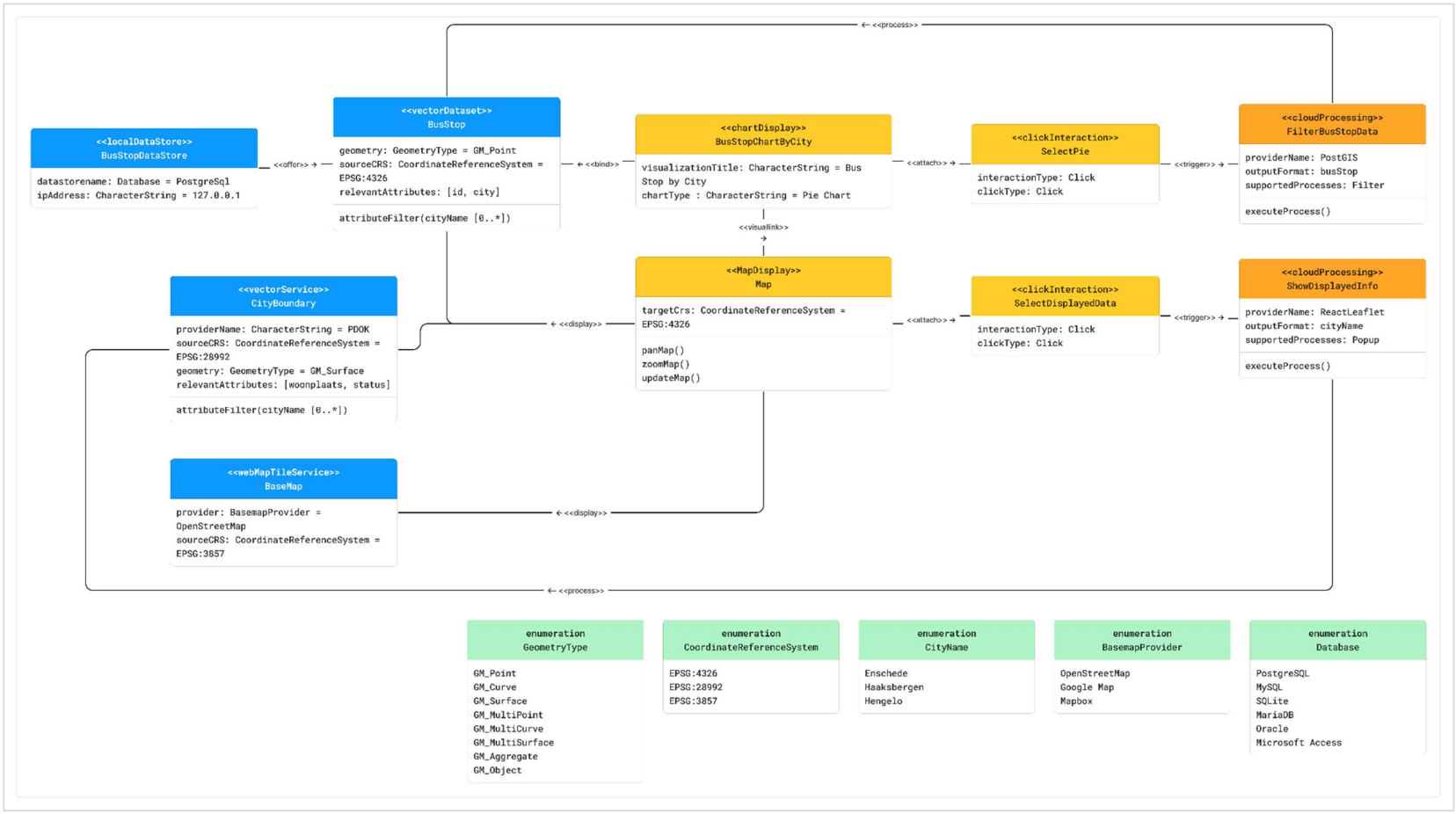


Figure 6. 2 The designed PIM diagram in the MDA implementation

6.2. PIM to PSM Transformation

This section demonstrate how the designed PIM diagram was transformed into PSM using the developed model transformation rules. This step was performed with reproducibility in mind. The most suitable target users to perform this step are:

- User Type #1: The Designers.
- User Type #2: The Developers.

For this model transformation, the developed transformation rules were followed:

- **Requirement #5.4.1.1**
  The PIM diagram was converted into PIM-JSON using the provided PIM-Profiles JSON file. In this implementation, only the pimModelCore.json was used to convert BusStopDataStore and BusStop PIM classes into its JSON schema using respectively LocalDataStore and Dataset JSON schema template.
- **Requirement #5.4.1.2 and #5.4.1.3**
  While most of the JSON schema follows the UML2JSON best practice document, the JSON pointers for indicating each class's stereotypes and operations followed the provided external JSON schemas.
- **Requirement #5.4.1.4**
  The association names were converted into association roles according to the reading direction. For example, the offer association becomes *offers* in the BusStopDataStore class and *isOfferedBy* in the BusStop class.
- **Requirement #5.4.2.1**
  The model transformation considered all attributes and operations including the inherited ones. For example, the LocalDataStore class instance BusStopDataStore inherited datastorename attribute which originally came from the parent class DataStore.
- **Requirement #5.4.2.2**
  Based on the selected stereotypes, several technology-dependent PSM-Profiles became unavailable to be picked, which then limited what PSM-Profiles to be used in the transformation. For example, the LocalDataStore class stereotype's *relatedPsmProfiles* attribute only allowed PostgreSQL, MongoDB, SQLite, MySQL, and MariaDB for the data store technology, which made options such as GoogleBigQuery and AmazonRedShift became unavailable. In this implementation, PostgreSQL was chosen.
- **Requirement #5.4.2.3**
  In this implementation, Express was chosen as the backend framework for the GWA. This decision determined that JavaScript became the programming language for the GWA.
- **Requirement #5.4.2.4, #5.4.2.5, #5.4.3.1, #5.4.3.2, and #5.4.3.3**
  A transformation script that mapped how each PIM attributes, operations, and associations transform into PSM class instances were created. In this implementation, an example of the transformation mapping can be seen in Figure 5.24 in earlier chapter.
- **Requirement #5.4.2.6**
  After the model transformation was performed and resulted the PSM-JSON file using the developed transformation script based on the previous Requirements, the resulted PSM-JSON file was opened manually to fill in the empty default values. For example, the resulted *Pool* PSM class

had its user, database, password, and port attributes emptied during the model transformation, which were then manually filled.

The result of this model transformation was the PSM-JSON file of a part of a complete PSM diagram shown in Figure 6.3. In this implementation, only BusStopData, Pool, PostGIS PSM classes and their associations that connect them to each other were part of this model transformation result. In addition, the FetchBusStop PSM class were also part of this model transformation result, but without any of its relevant associations.

6.3. PSM to Code Transformation

This section demonstrate how the designed PIM diagram was transformed into PSM using the developed model transformation rules. This step was performed with reproducibility in mind. The most suitable target users to perform this step are:

- User Type #1: The Designers.
- User Type #2: The Developers.

For this model transformation, the developed transformation rules were followed:

- **Requirement #5.5.1.1, #5.5.2.1, and #5.5.2.2**
  A transformation script that mapped how each PSM attributes, operations, and associations transform into source codes that were contained inside instances of source code files based on the proposed directory structure were created. In this implementation, an example of the transformation mapping can be seen in Figure 5.32 in earlier chapter.
- **Requirement #5.5.1.2**
  PSM associations were considered in generating the computer code of the GWA. In this implementation, an example of how associations are mapped into computer code can be seen in Figure 5.33 in earlier chapter.

The result of this model transformation was the computer code of a part of a complete GWA source code. In this implementation, only the *db.js* and buststopdata.js source code files were included in this model transformation results.

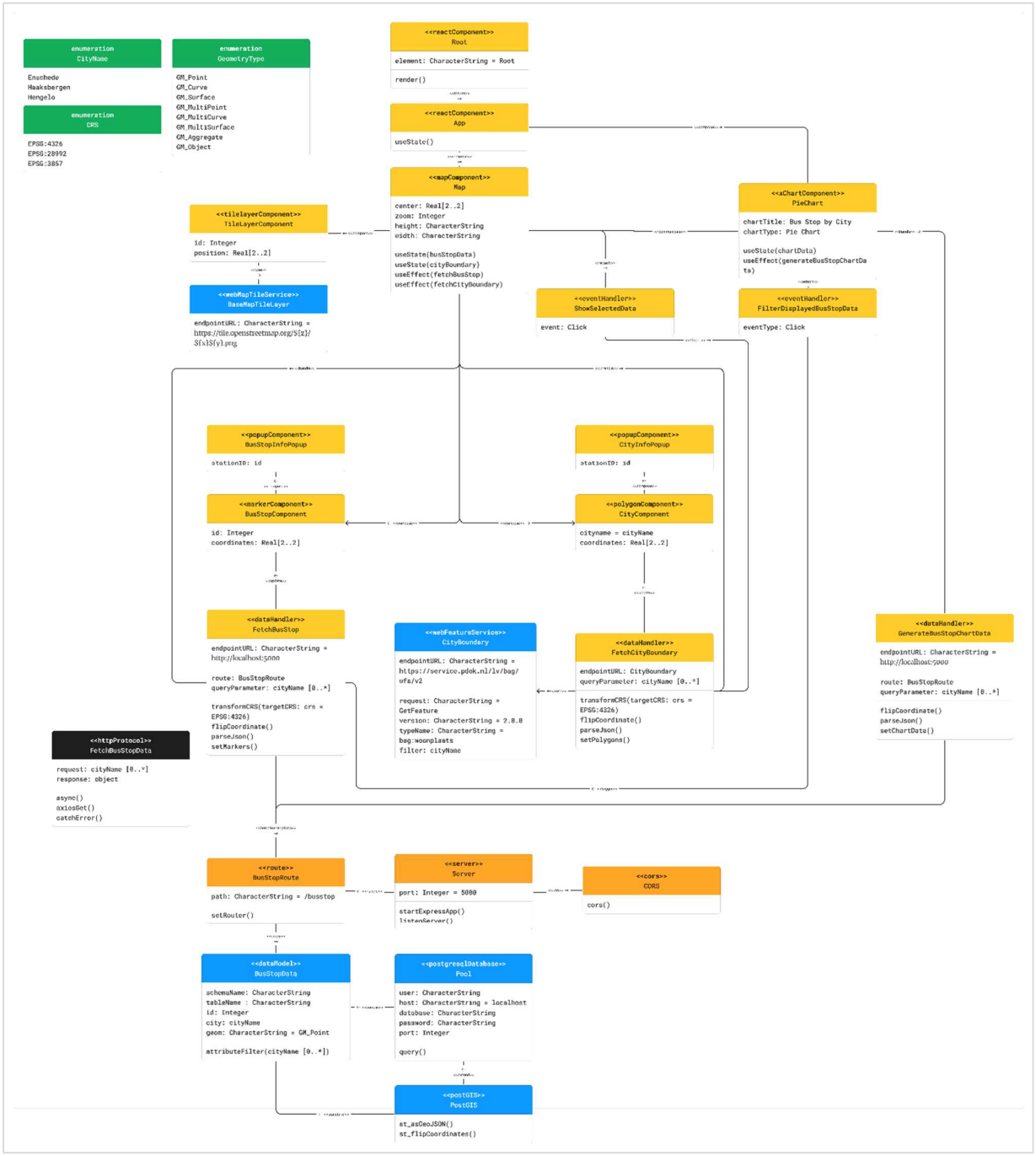


Figure 6. 3 The PSM in diagram format, resulted from the PIM-to-PSM transformation.

6.4. The Resulting GWA Source Code

This section compares between the resulted computer code from the MDA model transformations in this implementation and the source code of the GWA example. The result of the transformation and parts of the GWA source code were compared. The most suitable target users to perform this step are:

- User Type #1: The Designers.
- User Type #2: The Developers.
- User Type #3: The End Users.

In the Figure 6.4, the resulted db.js file was reviewed. It is shown that the the transformation result replicated the structure of the GWA example code except the fact that the example code used dotenv environment variable to store the access-related information. When the use of dotenv is deemed irrelevant to the implementation, the comparation shows that the model transformations successfully mapped information from PIM and the filled-in information during the transition between the transformations into the computer code.

In the Figure 6.5, the resulted busstopdata.js file was reviewed. It is shown that the transformation result were equivalent to the variables and if condition inside the try-catch statements. It is shown that the transformation result successfully mapped all information required to create a complete SQL query in the *query* variable. In addition, the transformed result left several parts of the if condition empty due to lack of information. This was because the information were mapped to come from another PSM classes which were not part of this implementation. This showcased how lack of information might caused incomplete code.

Figure 6. 4 Comparation of the resulted db.js file with GWA example code

Figure 6. 5 Comparation of the resulted busstopdata.js file with GWA example code

7. RESULT EVALUATION

This chapter evaluated the entirety of the proposed model-driven approach, including the developed UML profiles and the developed MDA-based development approach, using SWOT framework.

7.1. Strengths of The Proposed Approach

7.1.1. Simplified Development Approach for Non-Developer Users

The main value proposition of the proposed approach lies in the simplification development process that is friendly for non-developer users, specifically GIS and RS practitioners. The approach leverages pre-defined templates that users can customize to suit their use cases. For example, the PIM profiles serves as blueprints for The Designer users to build their own PIM diagram, which then users can choose the PSM profiles for transforming their PIM diagram into the selected technologies., which then leads to the generation of the computer code. There are only handful of steps the users have to go through to see the source code of their own GWA.

The developed approach ensures that it is convenient for any types of users to create their own GWA without the need of learning the traditional learning curve in developing a web application. The proposed approach uses standardized templates and predefined model transformations which ensure faster time and less effort in development and result in consistent GWA source code with less risk of errors as long as the users are within the documented directions and step like elaborated in this research.

7.1.2. Starting Point for Continuous Development by Developer Users

This research serves as a first stepping stone for the proposed development approach. The initial implementation offered a limited scope of development example, but it proved that the approach works to achieve a proper GWA development. The documented steps act as guidelines that can be expanded upon by developed users. For example, the developed UML stereotypes and profiles in this research can be tailored to better cater specific use case such as to create real-time geospatial dashboard for sensor data. This expansion possibility is beneficial because developers do not have to develop everything from zero and start with the documented approach from this research which reduces time and effort in development.

7.1.3. Structured Documentation of the Proposed Approach

The structured nature of the proposed approach makes it easy to follow even for those who might not have experience in web application development. The documented steps and processes allow users to systematically progress through the proposed development approach, from the PIM diagram building phase until the MDA transformations to turn the PIM into PSM and ultimately the computer code. For example, this research broke down the PIM instantiation step into 7 detailed requirements that users may follow to build their own PIM instance. The provided documentation promotes consistency and reusability of the proposed approach in developing many use cases of GWA.

7.2. Weaknesses of The Proposed Approach

7.2.1. Initial Complexity

It was a complex task to ensure the proposed approach is simple to be implemented by the targeted users. The development of the proposed approach involved rigorous planning and optimization tasks that consume time and require many trials and errors to refine. For example, the path to develop the first UML profiles for building PIM turned out to be a long steps of domain modelling and capturing the functional requirements for REST-based GWA, which then the UML profiles were subjected to modifications and refinements during the development of steps to perform PIM design. Even further refinements of the PIM profiles were performed when PIM-to-PSM transformations were developed. This complexity in developing the approach might lead to significant development cost and delays in project timelines, especially for developers who might intend to extend this proposed approach to cater their own specific use cases or domain.

7.2.2. Limited Flexibility

When the proposed approach only provides a limited number of technologies, it restricts the range of possible GWAs that can be developed. For example, the current PSM profile for mapping library indicates that there are only Leaflet and OpenLayers that users can choose from despite there are many other mapping libraries such as Mapbox GL JS and CesiumJS.

This flexibility limitation also covers the limited capabilities of the UML profiles in designing the PIM. If the stereotypes in the PIM profiles are too rigid or limited, they might not allow users to implement a specific GWA use case or functionality. For example, the *attributeFilter* operation contained in the VectorDataset class stereotype within the PIM profile allows datasets to be queried, but there has not been any class attribute or operations that indicate streaming data transformation is possible, which limits the GWA developed using the current version of the proposed approach to have that functionality.

7.2.3. Dependency and complexity of web application development frameworks and technologies

Developing web applications involves a myriad of technologies and frameworks, each with its own strengths, paradigms, and compatibility requirements. This dependency and complexity might pose significant challenges for the proposed approach when integrating various technologies in the GWA. One challenge is to ensure that different technologies work together seamlessly. For example, old frameworks like IBM CICS that based on COBOL, a language developed in the 1960s for mainframe computing, with modern front-end technologies like React or Axios can present compatibility issues.

7.3. Opportunities of The Proposed Approach

7.3.1. Adoption in GIS and RS community

GIS and RS communities are comprised of professionals and researchers who might want to showcase their spatial work with a web application. The adoption of the proposed development approach within the GIS and RS communities can become a significant help to achieve that goal. The proposed approach provides simplified steps for GIS and RS users to develop GWA for their projects and data, which allows the users to reach broader audience and increase the visibility of their work. The documented nature of the proposed approach also promotes consistency and quality in the GWA across the community which makes it easier for practitioners to collaborate.

7.3.2. Expansion of Use Cases

The proposed development approach can be extended to support a wide range of GWA development use cases across many domains like urban planning, earth science, environmental monitoring, and more. By catering to various domains, the approach may attract larger user base. This opportunity may also promote innovation and improvisation for the developed approach when it is being tested to demonstrate its capability to handle various requirements in those different domains.

7.3.3. Educational use

There is a significant opportunity for the proposed approach to be served as an educational tool for non-developer GIS and RS users in developing GWA with ease for their projects. The proposed approach breaks down web development tasks into manageable high-level design tasks which makes development process accessible for the non-developer users. Detailed documentation of the proposed approach may become a valuable educational resource on implementing model-driven development for their own use cases.

7.4. Threats for The Proposed Approach

7.4.1. Technological Obsolescence

As new tools and frameworks in web application development are being introduced, they open opportunities and innovation in GWA development. However, that also means that there is a challenge to keep up with the advancements to avoid obsolescence of the proposed approach. For example, some of the provided technologies in the PSM might have recent updates that change the syntax or replace a method with another new one. There may also situations where a technology in the provided PSM is not supported anymore. These kinds of changes have to be considered in the proposed approach.

7.4.2. Introducing New Learning Curve

While the proposed MDA-based GWA development approach aims to simplify the process, it does not eliminate the learning curve entirely. Instead, it shifts the focus from the traditional web development skills to understanding and utilizing new tools and concepts such as how to use the proposed UML profiles to build PIM, how to read and use the provided PIM-Profiles JSON file, and how to understand the GWA code resulted from all model transformations in the proposed approach.

7.4.3. Adoption resistance

Although the proposed development approach aims to simplify the process for GIS/RS practitioners, these users must still invest time in learning the basics of several concepts being used such as UML, MDA, and JSON. Despite efforts to make adoption easier, some users may find this involves steep learning curve.

Also, the proposed approach ultimately becomes one of many web application development techniques for users to choose from. For example, model-driven techniques like WebML and UWE exist, and they offer distinct benefits. Outside the domain of model-driven development, techniques such as component-driven development like Atomic Design offer different approaches that may appeal to users looking for a more granular approach to web application development.

8. CONCLUSION

This chapter concludes the research by summarizing all findings and discussions to answer all the research questions.

8.1. Research Outcome

The following subsections answer the sub-questions based on the research sub-objectives.

8.1.1. Conceptualizing REST-based GWA Functional Requirements

What are the common functions found from the GWA observations?

This question has been answered in length in Section 4.1 and 4.2. Based on 6 GWA observations performed in the Domain Modelling step, 21 common functionalities of GWAs were found.

1. Search for Location
2. Perform Processes
3. Open Other Resources
4. Authentication
5. Attribution
6. Get Other Layers
7. Initialization
8. Filter Data to Show
9. Show User's Current Location
10. Scale Control
11. Scale Bar
12. Get Info About Selected Location
13. Let User Add Data
14. Share Link
15. Download Data
16. Add Legend
17. Custom Right-Click Interaction
18. Use Different Units
19. Change Style
20. Interact With Currently Shown View
21. Move or Orient Current View

What are the observed functional requirements from the selected GWA functionalities?

This question has been answered and discussed in length in Section 4.3. A natural language-based descriptions of the observed functional requirements of several representative GWA functionalities (search for location, distance measurement, and select basemap layer), were elaborated.

The "search for location" functionality starts with the requirement that the users can enter and submit two types of location-based queries into a search bar in the client side. After the search input was submitted to the server, the server retrieved data from the data source and performed several queries and operations to the data. Lastly, the processed data were sent back to the client machine to update the presented interface with the search result.

The "distance measurement" functionality starts with an initiation by the user to start a measurement by clicking a button or context menu. After the initiation and relevant data input was received by the server, the creation of "job" to represent the measurement operation was conducted, which then accompanied the status of the measurement task until it was finished. The results of the measurement were sent back to client side to update the presented interface with the measurement result.

The "select basemap layer" functionality starts with the initialization of the GWA where a dedicated user interface component of the basemap options are presented. This initialization phase as well as the moment the user interacts with the options trigger a data exchange to the server where the required data or

metadata of the basemap layers were retrieved from the data source. The data were then sent back to the client side to update the presented interface.

The common concepts found in the three observations were the use of HTTP as the data exchange protocol and JSON as the data exchange format between client (frontend) and server (backend).

What are the potential class, attribute, operation, and relationship semantics found from the functional requirements?

This question has been answered and discussed in length in Section 4.4. Based on the noun/verb analysis, 8 potential UML classes were defined from the functional requirements. Potential attributes, operations, and associations were also extracted from the functional requirements which then assigned to the relevant classes. Below is the list of the potential semantics:

1. UserInteraction class
    - Attribute: interactionType
    - Operation: enter, interact, and initiate
2. Frontend class
    - Attribute: interfaceType
    - Operation: attach, display, construct, parse, extract, specify, process, update
3. HttpProtocol class
    - Attribute: interfaceType
    - Operation: attach, display, construct, parse, extract, specify, process, update
4. BackendServer class
    - Attribute: -
    - Operation: parse, extract, route, construct, create, update, validate
5. Job class
    - Attribute: uniqueIdentifier, status
    - Operation: handle
6. BackendService class
    - Attribute: input, output
    - Operation: execute, interact, retrieve
7. DataSource class
    - Attribute: -
    - Operation: -
8. Data class
    - Attribute: -
    - Operation: -

8.1.2. UML Profile Development

What are the proposed UML stereotypes for developing Platform Independent Model (PIM)?

This question has been answered and discussed in length in Section 5.1. The stereotypes for PIM development were categorized into three separate UML profiles based on MVC separation of concerns.

1. PimModelCore profile for data-related stereotypes.
2. PimViewCore profile for interface-related stereotypes.
3. PimControllerCore profile for data-interface mediatory stereotypes.

Within the PimModelCore, Dataset class stereotype represents the collections of data used in the GWA. The Dataset class stereotype is inherited by:

1. SpatialDataset class stereotype representing the collections of data that contains spatial property. This class stereotype is inherited by VectorDataset and RasterDataset class stereotype.
2. NonSpatialDataset class stereotype representing the collections of data that contains no spatial property.

DataStore class stereotype represents data to be used in the GWA. The DataStore class stereotype is inherited by:

1. LocalDataStore class stereotype representing data stores that reside on the local machine.
2. CloudDataStore class stereotype representing data store that is in some remote cloud environment.

DataService class stereotype represents service that provides access to data. The DataService class stereotype is inherited by:

1. SpatialDataService class stereotype representing data services that provide access to spatial data. This class stereotype is inherited by RasterTileDataService, VectorTileDataService, RasterDataService, and VectorDataService class stereotype.
2. NonSpatialDataService class stereotype representing data services that provide access to non-spatial data.

The PimModelCore has association stereotypes, each with different purpose:

1. Offer
2. Display
3. Process
4. Bind

Within the PimViewCore, UiComponent class stereotype represents the building block for user interface (UI) components. The UiComponent class stereotype is inherited by:

1. LayerSelection class stereotype representing a UI component that contains the layer options for a user to choose from.
2. Button class stereotype representing a clickable UI element that triggers an action.

3. SearchBar class stereotype representing a UI component specifically designed for users to enter search queries.
4. DataVisualization class stereotype representing represents UI components specialized in creating visual elements that communicate data insights. This class stereotype is inherited by MapDisplay and ChartDisplay class stereotypes.

UserInteraction class stereotype represents user interaction elements in a GWA. The UserInteraction class stereotype is inherited by:

- ClickInteraction class stereotype representing the action of clicking an UI component.

The PimViewCore has association stereotypes, each with different purpose:

1. VisualLink
2. Attach
3. Trigger
4. Display
5. Bind

Within the PimControllerCore, DataProcessing class stereotype represents the collections of data used in the GWA. The Dataset class stereotype is inherited by:

1. SpatialProcessing class stereotype representing processes that operate on spatial data.
2. NonSpatialProcessing class stereotype representing processes that do not involve spatial data.

Both SpatialProcessing and NonSpatialProcessing class stereotypes are inherited by:

1. LocalProcessing class stereotype representing processes that are accessed and handled in the local machine.
2. CloudProcessing class stereotype representing processes that are accessed and handled within cloud environment.

The PimControllerCore has association stereotypes, each with different purpose:

1. Trigger
2. Process

What are the proposed UML stereotypes for developing Platform Specific Model (PSM)?

This question has been answered and discussed in length in Section 5.2.

The stereotypes for PSM development were categorized based on the technology. Due to this structure, PSM stereotypes cannot be pinpointed like how PIM stereotypes were explained above. Three separate UML packages that act as the highest-level categorization based on the MVC separation of concern. Inside each MVC category are several UML packages that are based on the specific responsibilities

How do the proposed UML stereotypes be developed into UML profiles?

This question has been answered and discussed in length in Section 5.1, 5.2, and 5.3.

The PIM stereotypes were categorized into three separate UML profiles based on MVC separation of concerns.

1. PimModelCore profile for data related stereotypes (Model).
2. PimViewCore profile for interface related stereotypes (View).
3. PimControllerCore profile for data-interface mediatory stereotypes (Controller).

The PSM stereotypes were categorized with more complex mechanism than PIM stereotypes do. First, there are three separate UML packages that act as the highest-level categorization based on the MVC separation of concern. Inside each MVC category are several UML packages that are based on the specific responsibilities:

1. PsmModel package contains DataStore package representing the data storage technologies such as PostgreSQL, and DataService package representing services that provide data access such as WMS and WFS.
2. PsmView package contains FrontendFramework representing user interface libraries such as React, and MappingLibrary package representing web mapping libraries such as Leaflet.
3. PsmController package contains BackendFramework representing web server framework such as Express.JS, HttpClientLibrary representing HTTP client libraries such as Axios, and DataProcessing package representing geospatial processing services such as WPS.

8.1.3. Model Transformation Development and MDA Implementation

How do the proposed UML profiles support the development of the Platform Independent Model (PIM)?

This question has been answered and discussed in length in Section 5.3. The UML profiles for building PIM are used by users to build a PIM diagram. PIM instantiation requirements were developed to help users to build a PIM using the proposed approach. The requirements give directions on how users can build class instantiation using the selected PIM-Profile stereotypes, set a class name, set the attributes and operations, use enumeration for general use and for the specific GeometryType use, set associations, and annotate the associations properly.

What is the model transformation rules to transform the PIM to Platform Specific Models (PSM)?

This question has been answered and discussed in length in Section 5.4. After the PIM has been created, UML to JSON conversion is to be conducted to allow automated model transformation. PIM-to-PSM transformation rules or requirements were also developed to help users to transform their PIM into PSM using the proposed approach. The requirements give directions on how to convert UML into JSON, to map PIM elements into PSM elements, and to use the provided PIM-to-PSM transformation scripts.

What is the model transformation rules to generate the GWA computer code from the PSM?

This question has been answered and discussed in length in Section 5.5. After the PSM has been created by transforming PIM into specific PSM profiles, the PSM is then transformed into computer code based on the selected technologies. PSM-to-code transformation rules or requirements were developed to help

users to transform their PSM into computer code of the technology and frameworks they chose using the proposed approach. The requirements give directions on how to map PSM elements into the source code, and to use the provided PSM-to-code transformation scripts.

8.1.4. Result Evaluation

What are the strengths of the proposed approach and its resulting GWA?

This question has been answered and discussed in length in Section 7.1. The proposed approach has several strengths to bring upon:

1. Simplified Development Approach for Non-Developer Users
2. Starting Point for Continuous Development by Developer Users
3. Structured Documentation of the Proposed Approach

What are the weaknesses and limitations of the proposed approach and its resulting GWA?

This question has been answered and discussed in length in Section 7.2. The proposed approach has several weaknesses to consider:

1. Initial Complexity
2. Limited Flexibility
3. Dependency and complexity of web application development frameworks and technologies

What are the future opportunities and recommendations for the proposed approach?

This question has been answered and discussed in length in Section 7.3. The proposed approach has several future opportunities:

1. Adoption in GIS and RS community
2. Expansion of Use Cases
3. Educational use

What are the future challenges and threats for the proposed approach?

This question has been answered and discussed in length in Section 7.4. The proposed approach has several threats to consider:

1. Technological Obsolescence
2. Introducing New Learning Curve
3. Adoption resistance

8.2. Research Implication

The proposed development approach includes the developed UML profiles and the documented MDA-based development for generating REST-based GWA. This research result represents a step into a significant advancement which may provide benefits in GIS and RS domain.

The proposed development approach allows non-developer users, particularly those in the GIS and RS field, to design and build their own GWAs. They can focus on the high-level design aspects of their applications without the need of handling the coding complexities and the need of going through the traditional learning curve of web application development.

Additionally, the documented nature of the proposed development approach is also valuable for developers. It serves as a foundational framework that developers can build upon, extending its capabilities to support a broader range of technology stacks. As developers continue to enhance this proposed approach, it opens new possibilities for the types of GWAs that can be developed.

8.3. Research Limitation

The biggest limitation of the research is the time restriction that did not allow for achieving a full transformation of the properly running GWA in this project. The author also acknowledged his limit of knowledge and experience in this field which did not manage to execute a full-fledged MDA implementation with very few supervisions.

Another limitation is the few numbers of GWA observations performed during the domain modelling step. Adding more GWAs for observations add more validity to the collected and summarized functionalities that will ultimately result in the development of UML profiles. Another limitation is the few options for the platform-specific PSM profiles to be selected only allow a limited perspective on how the model transformations, the mapping of PIM elements into platform-specific PSM elements, and the mapping of PSM elements into specific computer code should be performed. When more use cases of how the MDA model transformations and the mapping of elements are considered, the proposed approach may be more mature.

Lastly, the author concluded that no matter how long time and how much considerations have been poured into building a UML class diagram or other class-based diagrams like UML profile of stereotypes, there will always be a better result to be achieved. While the author tried to do his best in developing the UML profiles and the proposed development approach based on the profiles within the provided time constraints, the author also agreed that in the hands of a true developer with experience in UML, this research may give better result.

8.4. Future Work and Recommendations

Several points can be suggested for future works that are based on this research. First, it is suggested to add more GWA for observation in the domain modelling phase for more representative collection of functionalities that can be ultimately modelled as UML profiles. In addition to it, adding more PSM profiles of technologies is also suggested for increasing the available options of GWA that can be developed using the proposed approach. This suggestion also leads to better UML profiles that can be developed.

It is also suggested for developing a more robust model transformation from PIM profile to PSM of choice. Currently in this research, the model transformations, and the mapping of elements between models were developed only within the limited perspective of what PSM profile options are available. If more PSM profile options are available, it can be included in the consideration of how PIM elements should be mapped and transformed.

Another suggestion for future works is the consideration of dependency between technologies. In the current research, the mechanism on how users can choose the PSM profile of technology to use is based

on what PIM class stereotypes were chosen. This neglected the possibility that a chosen PSM profile might influence what next PSM profile that can be chosen.

Lastly, future works should consider the continuous improvement of OGC API standards and the UML2JSON best practice document. Both are used in the proposed approach and are standards currently in development status. Improvements in these two standards may lead to significant improvements of the proposed approach.

Ameller, D., Burgués, X., Collell, O., Costal, D., Franch, X., & Papazoglou, M. P. (2015). Development of service-oriented architectures using model-driven development: A mapping study. *Information and Software Technology*, *62*(1), 42–66. https://doi.org/10.1016/J.INFSOF.2015.02.006

Barry, C., & Lang, M. (2003). A comparison of 'traditional' and multimedia information systems development practices. *Information and Software Technology*, *45*(4), 217–227. https://doi.org/10.1016/S0950-5849(02)00207-0

Betari, O., Filali, S., Azzaoui, A., & Boubnad, M. A. (2018). Applying a Model Driven Architecture Approach: Transforming CIM to PIM Using UML. *International Journal of Online and Biomedical Engineering (IJOE)*, *14*(09), 170–181. https://doi.org/10.3991/IJOE.V14I09.9137

Blanc, N., Cannata, M., Collombin, M., Ertz, O., Giuliani, G., & Ingensand, J. (2022). OGC API State of Play - A Practical Testbed for The National Spatial Data Infrastructure in Switzerland. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, *48*(4/W1-2022), 59–65. https://doi.org/10.5194/ISPRS-ARCHIVES-XLVIII-4-W1-2022-59-2022

Burgueño, L., Cabot, J., & Gérard, S. (2019). The future of model transformation languages: An open community discussion. *Journal of Object Technology*, *18*(3), 1–11. https://doi.org/10.5381/JOT.2019.18.3.A7

Conallen, J. (1999). *Building Web applications with UML*.

Distante, D., Pedone, P., Rossi, G., & Canfora, G. (2007). Model-driven development of Web applications with UWA, MVC and JavaServer faces. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4607 LNCS*, 457–472. https://doi.org/10.1007/978-3-540-73597-7_38/COVER

Elleuch, N., Khalfallah, A., & Ben Ahmed, S. (2007). Software architecture in model driven architecture. *ISCIII'07: 3rd International Symposium on Computational Intelligence and Intelligent Informatics; Proceedings*, 219–223. https://doi.org/10.1109/ISCIII.2007.367392

Fielding, R. T., Taylor, R. N., Erenkrantz, J. R., Gorlick, M. M., Whitehead, J., Khare, R., & Oreizy, P. (2017). Reflections on the REST architectural style and "principled design of the modern web architecture" (impact paper award. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, *Part F130154*, 4–14. https://doi.org/10.1145/3106237.3121282

Gellersen, H. W., & Gaedke, M. (1999). Object-oriented Web application development. *IEEE Internet Computing*, *3*(1), 60–68. https://doi.org/10.1109/4236.747323

Gómez, A., Rodríguez, R. J., Cambronero, M. E., & Valero, V. (2019). Profiling the publish/subscribe paradigm for automated analysis using colored Petri nets. *Software and Systems Modeling*, *18*(5), 2973–3003. https://doi.org/10.1007/S10270-019-00716-1/FIGURES/21

Helms, M. M., & Nixon, J. (2010). Exploring SWOT analysis – where are we now?: A review of academic research from the last decade. *Journal of Strategy and Management*, *3*(3), 215–251. https://doi.org/10.1108/17554251011064837/FULL/PDF

Home - PDOK. (n.d.). Retrieved July 1, 2024, from https://www.pdok.nl/

Hossain, M. I. (2023). Software Development Life Cycle (SDLC) Methodologies for Information Systems Project Management. *IJFMR - International Journal For Multidisciplinary Research*, *5*(5). https://doi.org/10.36948/IJFMR.2023.V05I05.6223

IEEE. (1990). IEEE Standard Glossary of Software Engineering Terminology. *Office*, *121990*(1), 1. https://doi.org/10.1109/IEEESTD.1990.101064

Jovanović, M., Starčević, D., & Jovanović, Z. (2013). Languages for model-driven development of user interfaces: Review of the state of the art. *Yugoslav Journal of Operations Research*, *23*(3), 327–341. https://doi.org/10.2298/YJOR121101007J

Kamatchi, Iyer, J., & Singh, S. (2013). Software Engineering:Web Development Life Cycle. *International Journal of Engineering Research & Technology*, *2*(3). https://doi.org/10.17577/IJERTV2IS3438

Klavans, J., & Kan, M. Y. (1998). Role of verbs in document analysis. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, *1*, 680–686. https://doi.org/10.3115/980845.980959

List of bus stops in the Netherlands | Datashop24.com. (n.d.). Retrieved July 1, 2024, from https://www.datashop24.com/product/List-of-bus-stops-in-the-Netherlands/

MDN. (n.d.). *HTTP Messages*. Retrieved June 13, 2024, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages

Meliá, S., Gómez, J., & Koch, N. (2005). Improving web design methods with architecture modeling. *Lecture Notes in Computer Science*, *3590*, 53–64. https://doi.org/10.1007/11545163_6/COVER

OGC. (n.d.-a). *Best Practice for OGC - UML to JSON Encoding Rules*. Retrieved June 15, 2024, from https://geonovum.github.io/uml2json/document.html#toc20

OGC. (n.d.-b). *OGC API*. Retrieved June 6, 2024, from https://ogcapi.ogc.org/

OGC. (n.d.-c). *OGC API - Common - Part 1: Core*. Retrieved November 19, 2023, from https://docs.ogc.org/is/19-072/19-072.html

OGC. (n.d.-d). *OGC API - Environmental Data Retrieval Standard*. Retrieved June 6, 2024, from https://docs.ogc.org/is/19-086r6/19-086r6.html

OGC. (n.d.-e). *OGC API - Features - Part 1: Core corrigendum*. Retrieved June 6, 2024, from https://docs.ogc.org/is/17-069r4/17-069r4.html

OGC. (n.d.-f). *OGC API - Maps - Part 1: Core*. Retrieved June 6, 2024, from https://docs.ogc.org/DRAFTS/20-058.html

OGC. (n.d.-g). *OGC API - Processes - Part 1: Core*. Retrieved June 6, 2024, from https://docs.ogc.org/is/18-062r2/18-062r2.html

OGC. (n.d.-h). *OGC API - Records - Part 1: Core*. Retrieved June 6, 2024, from https://docs.ogc.org/DRAFTS/20-004.html

OGC. (n.d.-i). *OGC API - Tiles - Part 1: Core*. Retrieved June 6, 2024, from https://docs.ogc.org/is/20-057/20-057.html

OGC API - Common - Overview. (n.d.). Retrieved June 25, 2024, from https://ogcapi.ogc.org/common/overview.html

OMG. (2014). MDA Guide rev 2.0. In *OMG Document ormsc/2014-06-01*. https://www.omg.org/cgi-bin/doc?ormsc/14-06-01

Pop, D. P., & Altar, A. (2014). Designing an MVC Model for Rapid Web Application Development. *Procedia Engineering*, *69*, 1172–1179. https://doi.org/10.1016/J.PROENG.2014.03.106

Rivero, J. M., Grigera, J., Rossi, G., Robles Luna, E., & Koch, N. (2012). Towards Agile Model-Driven Web Engineering. *Lecture Notes in Business Information Processing*, *107 LNBIP*, 142–155. https://doi.org/10.1007/978-3-642-29749-6_10

Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. C., Canali, L., & Percannella, G. (2016). REST APIs: A large-scale analysis of compliance with principles and best practices. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9671*, 21–39. https://doi.org/10.1007/978-3-319-38791-8_2/FIGURES/4

Royce, W. W. (1970). MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS. *Proceedings of the 9th International Conference on Software Engineering*. https://doi.org/10.5555/41765.41801

Singh, Y., & Sood, M. (2010). The Impact of the Computational Independent Model for Enterprise Information System Development. *International Journal of Computer Applications*, *11*(8), 975–8887.

Soni, A., & Ranga, V. (2019). API Features Individualizing of Web Services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*. https://doi.org/10.35940/ijitee.I1107.0789S19

Stack Overflow. (n.d.). *Stack Overflow Developer Survey 2023*. 2023. Retrieved July 1, 2024, from https://survey.stackoverflow.co/2023/#worked-with-vs-want-to-work-with-database-worked-want-prof

Stack Overflow Trends. (n.d.). Retrieved July 1, 2024, from https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs%2Cvuejs3

Thomas, D. (2004). MDA: Revenge of the modelers or UML Utopia? *IEEE Software*, *21*(3), 15–17. https://doi.org/10.1109/MS.2004.1293067

Wang, Y. (2016). Semantic information extraction for software requirements using semantic role labeling. *Proceedings of 2015 IEEE International Conference on Progress in Informatics and Computing, PIC 2015*, 332–337. https://doi.org/10.1109/PIC.2015.7489864

Whittle, J., Hutchinson, J., & Rouncefield, M. (2014). The state of practice in model-driven engineering. *IEEE Software*, *31*(3), 79–85. https://doi.org/10.1109/MS.2013.65