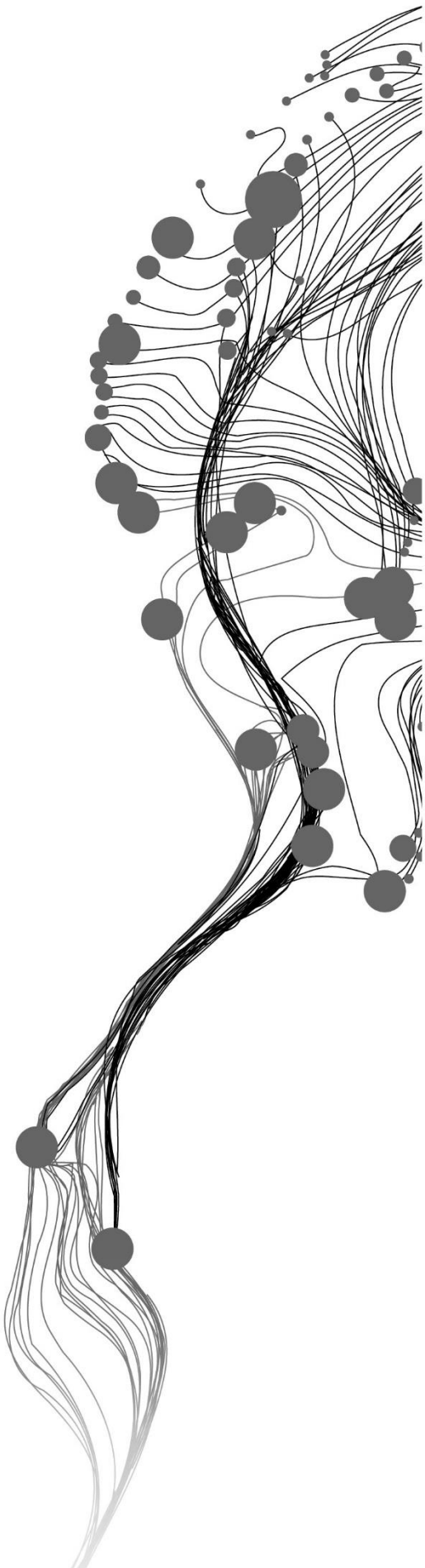
ARCHITECTURAL PATTERNS FOR DEVELOPING GEOSPATIAL WEB AND MOBILE APPLICATIONS

MORTEZA YAGHOUBKHANI GHIYASVAND
August 2021

SUPERVISORS:
Dr. J.M. Morales
Dr. Ir. R.A. de By



ARCHITECTURAL PATTERNS FOR DEVELOPING GEOSPATIAL WEB AND MOBILE APPLICATIONS

MORTEZA YAGHOUBKHANI GHIYASVAND
Enschede, The Netherlands, August 2021

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
Dr. Ir. R.A. de By

THESIS ASSESSMENT BOARD:
Prof. dr. R. Zurita Milla, Name (Chair)
Prof. dr. ir. P.J.M. van Oosterom (External Examiner, Name Institute)

ABSTRACT

The rise of the information communication technology (ICT) and infrastructure, the recent technological advancements in computational power, and the introduction of new tools such as smartphones, tablets, and laptops, are giving service providers and software developers a unique opportunity to provide more and more functionalities and services through the web. Thanks to this opportunity, like others, GI scientists and professionals use the web as a new infrastructure to provide GIS services. Today almost all desktop GIS functionalities could be provided on the web using geospatial web applications. A geospatial application uses web technologies to obtain data and information, process this data, and disseminate it. It also uses the web for communicating between these three main components.

Compared to desktop GIS, geospatial web applications have several advantages, and the most important one is that their development process is cheaper and faster. There is no need for updating software and data in geospatial web applications. These applications have higher levels of accessibility to users, and they are better suited to data acquisition and dissemination on the network. However, there are some issues in developing geospatial web applications that cause some disadvantages. Along with technological advancement, users expect more functionalities from geospatial applications. Users are always looking for more advanced analysis on massive geospatial data (with various data types from different resources) in less time. That is why geospatial web applications should be able to access and handle massive amounts of spatial and non-spatial data with different formats from multiple sources. Another major issue in geospatial web application development is rapid changes in web development technologies and platforms, making it hard for the applications to keep up with the pace. Since most of these applications are based on a particular technology, they cannot be updated for the new technologies and have to be developed from scratch. Also, issues regarding application security and spatial privacy are among the most significant geospatial web application development challenges.

Several web application development methodologies such as Model-Driven Development (MDD) or Data-Driven Development could help developers deal with these issues. This research proposes the Pattern Based Model Driven Architecture (PBMDA) for geospatial web application development. The PBMDA approach is based on the Model Driven Architecture (MDA) and tries to integrate software design patterns into the MDA process. MDA is based on the MDD notation and emphasizes the use of models with different abstraction levels in all phases of the web development process, and software design patterns are standard solutions for recurring problems in software development. Using the PBMDA approach, we can ensure flexibility, interoperability, better communication, and inclusion in the web development process.

Based on the PBMDA approach, we first developed a Domain Specific Language (DSL) for a particular geospatial web application. Then, considering the technical specifications related to the application's implementation and preferred platforms (implementing using a JavaScript stack, e.g., MongoDB, ExpressJS, ReactJS, and NodeJS), we created a Platform Specific (PSM) metamodel. We generated a PIM-PSM transformation function using pattern-based model transformation (PBMT) with PIM and PSM metamodels. Next, the Computation Independent Model (CIM) has been developed using the application logic. Using created CIM and generated DSL (PIM metamodel), we have developed the PIM. Finally, using the PIM and transformation function, we have generated the PSM. The application code generated automatically from the PSM.

Keywords:
Geospatial web application, Model Driven Architecture (MDA), Pattern Based Model Driven Architecture (PBMDA), model, metamodel, Platform Independent Model (PIM), Platform Specific Model (PSM), Domain Specific Language (DSL), software design patterns, the Observer design pattern

ACKNOWLEDGEMENTS

This research would not have been possible without the support of many people. Many thanks to my first supervisor Dr. Morales who read my numerous revisions and helped make some sense of the confusion. Also, thanks to my second supervisor Dr. de By, who offered guidance and support.
Finally, thanks to my family and friends who endured this long process with me, always offering support and love.

TABLE OF CONTENTS

Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREBVIATIONS

API – Application Program Interface
CIM – Computation Independent Model
DSL – Domain Specific Language
EMF – Eclipse Modeling Framework
GoF – Gang of Four
JSON – JavaScript Object Notation
MDA – Model Driven Architecture
MDD – Model Driven Development
MDE – Model Driven Engineering
MDSE – Model Driven Software Engineering
MOF – Meta Object Facility
OGC – Open Geospatial Consortium
OO – Object Oriented
OOP – Object Oriented Programming
OMG – Object Management Group
PBMDA – Pattern Based Model Driven Architecture
PIM – Platform Independent Model
PSM – Platform Specific Model
QVT – Query/View/Transformation
QVTc – Query/View/Transformation operational core
QVTr – Query/View/Transformation relational mapping
QVTo – Query/View/Transformation operational mapping
REST – Representational State Transfer
Relational Unified Process – RUP
SDI- Spatial Data Infrastructure
SOAP – Simple Object Access Protocol
UML – Unified Modeling Language
WMS – Web Mapping Service
WFS – Web Feature Service
WWW – World Wide Web
XMI – XML Metadata Interchange
XML – Extensible Markup Language

1. RESEARCH INTRODUCTION

1.1. Introduction / Research background

Currently, thanks to recent technological developments in information technology, expansion of internet networks (e.g., 3G, 4G, and 5G in the near future) to almost any place in the world, and recent improvements in smartphone technology, web, and mobile applications have become sophisticated tools. They are used in many aspects of our daily life. From personal uses such as car navigation, shopping, and package delivery to highly elaborated uses in businesses to disseminate valuable information and data through different platforms to diverse stakeholders in big companies and governments.

A web application (or web app) is software that, unlike other computer software that runs on the local device, runs on a web server (Chaffee, 2000). Web applications are complicated systems dependent on a variety of software and hardware tools, internet protocols, programming languages, user interfaces, and many more standards. In many cases, these web applications have geospatial aspects and provide some spatially related services. In some other cases there are geospatial web applications which their primary goal is to provide explicit spatial services for end users (professionals who need this geo-related information).

Recent developments in information networks, along with the exponential increase in computing power, and the incredible developments in smartphones, tablets, and other location-aware devices, led to unprecedented growth in the number of geospatial web applications. However, most of these applications are just viewers and mush-up applications with some innovative look and feel, but they follow the traditional paradigm of working with their data. There are several problems with this kind of web application: the high cost of production, selling, and maintenance, being time-consuming and inefficient in many cases, and tightly coupled data and services.

Current geospatial web applications face serious productivity issues in terms of required time and labor for application development. The final product could be late for the market, and in some cases, essential functionalities might be missed by the development team. This problem is mainly because of a lack of documentation and proper communication in web development teams since each team uses a different approach, and it is hard to communicate their work with new team members, which leads to problems in application maintenance and upgrade. The high dependency on a specific platform and technology also is a problem with current approaches to web development. This causes severe issues when a new technology is introduced, it is almost impossible to upgrade the application, and it should be developed from scratch. Usually, geospatial web applications need to have access to data and web services from multiple resources. They should be able to integrate these different web services and datasets and provide the desired functionalities for users. However, most current geospatial web applications are tightly coupled with certain types of data (specific data with specific format from a particular data source) and technology, which is considered a limitation since, with a change in data and technology, they cannot function well and provide necessary services.

With continuous enhancement of quality and quantity of web technologies and web development platforms, several web application frameworks and design patterns have been developed to solve some of the problems and issues with traditional web application design patterns. This research emphasizes creating a clear architectural pattern that system developers can use to design and implement cutting-edge geospatial web and mobile applications using the Model-Driven Architecture (MDA) development process. The MDA is an architectural framework for software development adopted by the Object Management Group (OMG) in 2001. In the MDA approach, the model is the central element in the process of software development, and model implementation and software code generation happen through a set of automatic transformation rules.

Software engineers and web developers are increasingly using the MDA to develop all kinds of software and web applications for different domains. There is a logical separation between the application's conceptual model and

its implementation on a specific platform with a particular technology in the MDA. This separation of concerns makes this methodology ideal for developing geospatial web applications. Plus, the fact that MDA is based on models and modeling processes makes it more unique for geospatial application development since these models can help experts capture and express geospatial domain problems and their respective solutions more efficiently. Working with computer modeling languages (such as Unified Modeling Language – UML) provides a unique capacity for MDA in documenting application development processes. We also try to integrate the software design pattern concept into the MDA by proposing the Pattern Based Model Driven Architecture (PBMDA) approach.
The proposed methodology for web application development in this research will have the following characteristics: it is based on the principles of separation of concerns, provides the capacity for code and model reusability, can be used for developing new applications and models, needs less time for application development (especially for new applications that are based on it), and has a lower level of code complexity.

1.2. Problem statement

There have been several kinds of research on applying the MDA framework to web application development. The MDA framework is currently used by many big companies and organizations to increase the quality of software and web applications in their respective field. Design patterns are also widely used by software engineers and web developers as a development strategy in the software industry and several research projects. However, the number of research articles and applications that combine these two software development paradigms is restricted to a few workshop articles, scientific reports, and a limited number of development projects.
This research is trying to develop the PBMDA, which essentially is an architectural pattern that integrates design patterns into the MDA for geospatial web applications. It makes it unique in terms of the research approach and implementation. The resulting web application, which is expected to be created by the end of this research, will be one of the few (if not the only) examples of applying the PBMDA approach for geospatial web application development.

1.3. Research objectives

This research's main objective is to define a clear architectural pattern for the design and development of cutting-edge geospatial web and mobile applications. This architectural pattern facilitates the integration of software design patterns into the MDA approach to improve the development process of geospatial web applications in terms of application quality and the amount of time and energy required for application development and maintenance. The proposed PBMDA provides established solutions for recurring geospatial application design problems. It could be used by GIS developers and software engineers to develop an elaborated geospatial web application based on scientific methods.
Based on the main objective, there are four sub-objectives to this research:

1- To investigate different web development methodologies

2- Introducing MDA methodology for geospatial web development

3- To propose a Pattern-Based Model Driven Architecture for geospatial web application development by integrating software design patterns into MDA

4- To develop a prototype geospatial web application based on the proposed pattern to demonstrating its applicability to the field

1.4. Research questions

Related questions to the first objective:

- What are the main approaches and methodologies for web application development?
- What are the essential criteria to consider for web application development?
- What is the best methodology for web application development?
- What are the specific requirements of geospatial web applications that should be considered in the development process?

Related questions to the second objective:

- What is MDA, and what are its main characteristics
- How this approach addresses those criteria important for geospatial web application development?
- What are the main steps to develop a geospatial web application using the MDA approach?

Related questions to the third objective:

- What are design patterns, and how they can improve the web development process?
- What are the main challenges in geospatial web application development using MDA?
- How can software design patterns be integrated into MDA to create a Pattern-Based Model Driven Architecture (PBMDA)?

Related questions to the fourth objective:

- What are the user requirements in a proposed geospatial web application?
- What are the main functionalities (components) in this application to address user requirements?
- How can we use the PBMDA approach to develop a web application for one of these functionalities?

1.5. Research outline

The main phases of this research are literature review, methodology development, and implication of the developed methodology, conclusion, and discussion. Here you could find the structure of this thesis:

Chapter 1 provides the necessary information for the research, such as its background, problem statement, and research objectives. Research questions related to each objective are also represented in this chapter.

Chapter 2 is on the literature review and is the primary basis for the research. At first, the notion of Object-Oriented software development and its central concepts are reviewed. The next part of the literature review contains an overview of web application development, its history, and its current status. In this part, the current web applications and their development problems are examined, and a set of criteria for web application development focusing on geospatial web applications is presented. In the end, we examine different web application development methodologies and discuss the advantages and disadvantages of each methodology concerning application development criteria.

Chapter 3 is on the research methodology development. At first, the MDA methodology for software development and its main characteristics are explained. Then we discuss how MDA could be used for developing web applications. The rest of this chapter is about the PBMDA methodology and how it could integrate software design patterns into the MDA process. At the end of this chapter, based on the PBMDA, a platform-independent

metamodel and a platform-specific metamodel for a prototype geospatial web application will be developed. These metamodels are the basis for developing a geospatial web application in the next chapter.
In **chapter 4**, we are using the PBMDA methodology and the two metamodels (platform-independent metamodel and platform-specific metamodel) acquired in chapter 3 to develop a geospatial web application (an application to locate and find properties based on user queries). To develop this prototype application, we go through all stages of application development in PBMDA and generate CIM, PIM, PSM, and finally, the application code.
In **chapter 5**, we try to answer research questions based on the results of previous chapters. In this chapter, we also try to summarize the entire research and what has been done in each chapter. This chapter also includes sections about research limitations and recommendations for future research on this topic.

2. LITERATURE REVIEW

2.1. Introduction

The main goal of this chapter is to provide an overview of the current and the past academic literature on the main topics of the research. It also provides a basic overview of Object-Oriented (OO) software development and focuses on how OO software development, software design patterns, and MDA can interact. The outputs of this chapter are the main base for the next chapter, which is about developing a methodology for PBMDA and how to implement it for a geospatial web application.

At first, the Object-Oriented Programming (OOP) paradigm in computer science will be introduced, and its main principles will be discussed. The notion of OOP is of high importance to our research and having a good understanding of the principles of the OO is vital in software development. Moreover, OO is one of those concepts in computer science directly related to other central topics in this research, such as software design patterns and the MDA.

After explaining the OOP and its principles, software design patterns and their importance in software development will be explained. Finally, there will be some explanation of the related definitions, history, and their importance in developing software and geospatial web applications.

The next part of this chapter reviews web applications and web application development methodologies, tools, and frameworks, emphasizing geospatial web applications and their unique characteristics.

The last part of this chapter is devoted to MDA as one of the software and web application development methodologies. After describing the main concepts and principles of MDA, the main steps toward developing software using MDA will be examined.

In research on geospatial web applications, the first thing to do is provide a clear definition of the main research concepts and keywords. One of the main keywords in this research is "geospatial web application," so in rest of this chapter will discuss the current definitions and explanations on this keyword.

2.2. Object oriented software development

The OO paradigm in software development is used to capture/manage the complexity of real-world problems using the principles of abstraction and the notion of objects and classes to encapsulate this knowledge (Wirfs-Brock et al., 1990). So, identifying the proper object and classes, their attributes and methods, and relationships between them is among essential goals in the OO software development.

2.2.1. Object oriented software development life cycle

While there are several methodologies in the software development domain which use the OO paradigm (Brambilla et al., 2012; G. Rode, 2008; Wirfs-Brock et al., 1990) there is not a widespread agreement in the academic and professional community on the common standards and specifications related to it and its implementation.

In general, regardless of the methodology in use to implement the OO approach in software development, it can be concluded that based on the OO paradigm, the software development life cycle generally consists of four main phases: analysis, design, implementation, and testing (Armstrong, 2006; G. Rode, 2008; Wirfs-Brock et al., 1990). Of course, there might be some overlapping or iteration in these phases based on the specific OO methodology in use, but most of them more or less include these four phases.

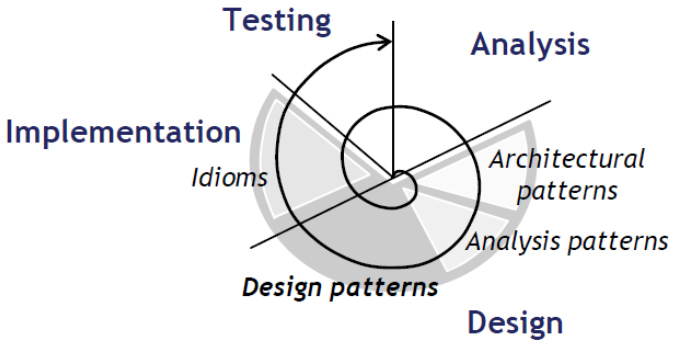


Figure 2-1- The OO software development life cycle and patterns (G. Rode, 2008)

As is visible in figure 2-1, different kinds of patterns could be used in different phases of software development. Architectural patterns have the highest level of abstraction and large design granularity. They are used in the early design phases of the software development life cycle. Analysis patterns have a medium granularity, and they usually are dealing with domain issues. Design patterns have a small granularity and are in a lower abstraction level. They also are used in the design phase and are closely related to the implementation phase. With the smallest granularity, idioms fall entirely in the implementation phase. They directly deal with a specific programming language (G. Rode, 2008; Vlissides, 1997).
The software design phase is central to the OO approach and should be sound and robust. Software engineers and designers use architectural patterns, analysis patterns, and design patterns for a faster, more scalable, and better software and application design.

2.2.2. OO software development concepts

Although several books, scientific articles, professional reports, and conference papers exist about the OO software development paradigm, there is still a lack of agreement on its fundamental concepts between different parties and individuals in the software engineering domain. However, several computer scientists and professionals in the software community tried to develop such a concept that others could use widely (Wirfs-Brock et al., 1990).
In one of the best practices, Armstrong identified thirty-one important OO-related concepts in the software development field based on a survey of several existing publications about OO software development. He found that only eight of those concepts are utilized in nearly every classification (Armstrong, 2006). These eight fundamental concepts of the OO paradigm identified by several academics and professionals are shown in the table 2-1.

Construct	Description
Structural construct	
Abstraction	Creating classes to simplify aspects of reality using distinctions inherent to the problem
Class	A description of the organisation and actions shared by one or more similar objects
Encapsulation	Designing classes and objects to restrict access to the data and behaviour by defining a limited set of messages that an object can receive.
Inheritance	The data and behaviour of one class is included in or used as the basis for another class.
Object	An individual, identifiable item, either real or abstract, which contains data about itself and the descriptions of its manipulations of the data.
Behavioural construct	
Message	A way of access, set, or manipulate information about an object
Message passing	An object sends data to another object or asks another object to invoke a method
Polymorphism	Different classes may respond to the same message and each implement it appropriately.

Table 2-1- Armstrong's two-construct OO taxonomy - adopted from (G. Rode, 2008)

The notion of objects and classes provides the opportunity to encapsulate data and behavior, while inheritance allows new classes based on the existing classes (Armstrong, 2006). Messaging is the interaction between two objects, and the polymorphism concept makes it possible that different objects deliver different responses to the same message.
All these main concepts are interconnected, and their integration provides the basis for OO software development. Also, the OO software development concepts classification by Gamma et al. 1995 (which is the primary resource for the classification of design patterns in this thesis) includes these main concepts too. Using these concepts by software engineers and developers promotes the software development process and its need for abstraction, flexibility, loose coupling, and reuse (G. Rode, 2008; J. Rode et al., 2005) in the process.
This classification somehow matches with the Gang of Four (GoF) classification of design patterns (Gamma et al., 1995), where software design patterns are categorized into two main categories, namely structural, behavioral, or creational. This classification is also in good harmony with different diagram types in the UML, targeting structural and behavioral characteristics of software systems (Baresi et al., 2001). Based on the primary goal of the research and for simplification and easier implementation, we will use this classification of the main object-oriented concepts and try to relate all the other concepts to them.

2.2.3. OO software development: analysis

The term analysis in the OO software development paradigm is always concerned with understanding the needs and requirements of the software system (G. Rode, 2008). During the analysis process, software engineers determine the system's needs and requirements and what must be done to satisfy those needs and requirements.
During the OO software development analysis phase, all the objects and classes in the development process and concepts like abstraction and relationships must be identified and expressed in a conceptual model (G. Rode, 2008). The result of this phase (the analysis phase) is always represented in the form of a conceptual object model (Larman, 2004) usually expressed using UML diagrams.

2.2.4. OO software development: design

The design phase is the second phase in the overall process of OO software development. It determines how things should be done to satisfy the system's specific requirements (Schmidt, 2006; Wirfs-Brock et al., 1990). As it was discussed earlier, the conceptual model developed in the analysis phase identifies main system requirements and related objects and classes without considering any specific software solution or technology. On the other hand, the OO model resulting from the design phase describes software entities, objects, classes, and technologies required to achieve such functionalities. The design phase includes all the required steps for defining and implementing required objects and classes within a particular software. The OO design phase creates an integrated model to satisfy the system's particular needs and requirements introduced in the analysis phase (Larman, 2004; Wirfs-Brock et al., 1990). The software system in this phase consists of a complex combination of software objects, classes, and methodologies for different abstraction levels.
There is no general agreement on the fundamental OO design concepts and methodologies. However, there are several methodologies and approaches for software design based on the OO paradigm (e.g., MDA (OMG, 2003) and Relational Unified Process – RUP (Kruchten, 1999)). However, none of them is widely accepted and used in the scientific community and software development industry (Capretz, 2003). That is why the implementation of the design phase (e.g., software object descriptions, their responsibilities, and the relationship between them and with the other parts of the system) mainly depends on the choice of software engineer and developer to use a specific OO methodology (G. Rode, 2008).
In the OO design phase, we could use different patterns to make the design process more efficient, better documented, and easily communicable between team members and stakeholders. These patterns (namely architectural patterns, analysis patterns, and design patterns) have various granularity and abstraction levels. They

are used by software engineers and developers during different phases of the design process. The following sections cover in more details these patterns.

2.2.4.1. Architectural patterns

Architectural patterns help us define the main structure of a software system, including the notion of different sub-systems and how they interact (e.g., how they communicate and collaborate). Architectural patterns have the highest level of abstraction between the other patterns in software development (analysis patterns and design patterns). They are applicable for specific scenarios in software development, not in all of them (Buschmann et al., 1996).

One important point regarding architectural patterns is that they cannot represent the whole software system, and their primary role is in the design phase. Usually, architectural patterns should be applied in the design phase and be used with other patterns (i.e., design patterns) to fill the gaps and better represent the software system (G. Rode, 2008; J. Rode et al., 2005).

2.2.4.2. Design patterns

In the definition of design patterns represented by GoF, there is more emphasis on the problems happening in the software design phase, directly related to the general OO paradigm (G. Rode, 2008). However, Borchers (Borchers, 1999) provides a broader definition for software design patterns: "A software design pattern is generally considered to be a proven solution of a recurring software engineering problem that balances the competing design constraints optimally for a certain type of situation."

While architectural patterns play an essential role in the software design process and should be selected based on the project needs and specifications, design patterns developed to be used autonomously (Buschmann et al., 1996) in software development. However, some design patterns are more suitable to be used with specific architectural patterns.

2.2.5. OO software development: implementation

OO software development paradigm can be considered as the integration of specific development phases such as analysis, design, and implementation. In the implementation phase, the software system analyzed and designed in previous stages is translated into the software code in the desired programming language or using appropriate software or tool. In addition, other system requirements such as user interfaces and datasets should be prepared and incorporated into the main software in this phase.

2.2.6. Unified modeling language (UML)

UML which considered by many as the lingua franca in software engineering, is an OO-based modeling language (Wimmer et al., 2007). Using UML, we can express each aspect of any software development project in the form of models. Furthermore, it uses various diagrams to graphically represent each model (Schwinger & Koch, 2006). Thus, UML can be used in all phases of life cycle of any software development project, regardless of the tools and techniques involved in the project.

UML provides a general-purpose, standardized modeling language by integrating business models and data modeling techniques into the OOP paradigm. There are two major diagram types in UML: structural diagrams and behavioral diagrams. Structural diagrams are used to show the static structure of a software system. Behavioral diagrams capture and represent the system's behavior, showing the dynamics and changes in all the entities and objects. The following diagram shows the UML and its different diagrams based on the above classification (Alesheikh et al., 2002).

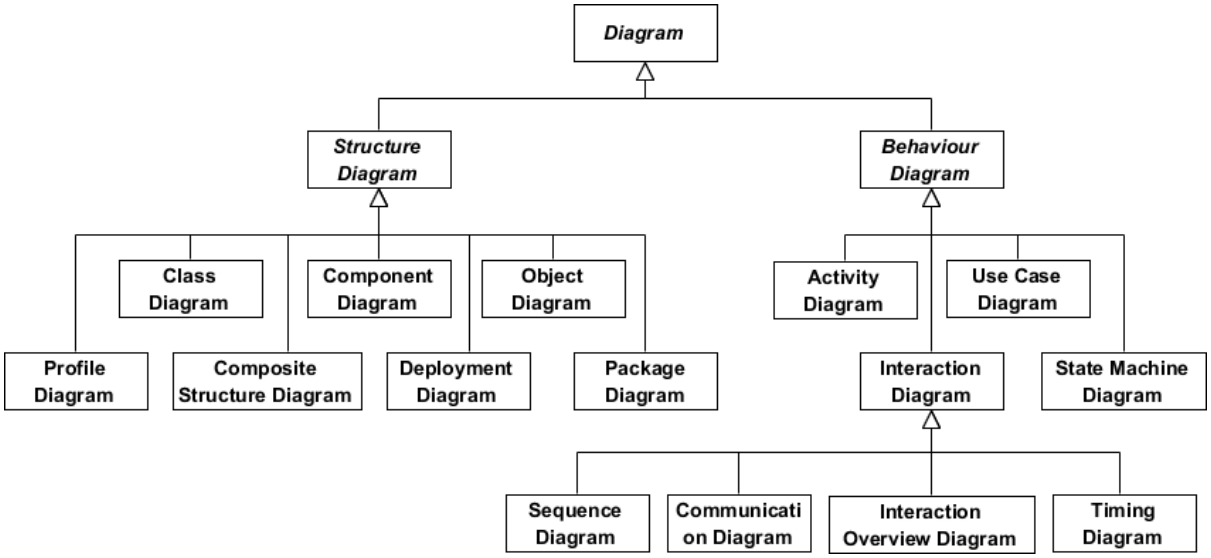


Figure 2-2- UML diagrams

Models in UML are expressed using different diagrams: for example, an object model could be expressed using the object diagram, a dynamic model could be expressed using a Sequence diagram, and a functional model could be expressed with a use Case diagram (Object Management Group, 2017). UML models are valuable tools for software engineers to represent OO patterns because they are very good at illustrating objects and classes. In the case of software design patterns, although GoF's design patterns predate UML and other forms of graphical representation were used to express them, UML is the primary tool today used to express design patterns because of its unique characteristics.

One of the benefits of using UML in OO software development is that it can be used in different phases of the software development process. In UML, the software system could be presented using models (e.g., functional model, object model, and dynamic model).

UML traditionally has been used as a metamodel, and software engineers and developers used it to create models. There is a mechanism in UML called the profile mechanism. By creating definitions for object notations and nomenclatures, the profile mechanism allows us to extend metaclasses in the UML and change them according to every project purpose.

However, there are few critics of the UML and its use. The first one is that the language is most suitable and easy to use for modeling purposes in the IT domain and there are some limitations in using it for other domains such as finance, biology, and geoinformation science (Thomas, 2004). The other issue is that the language is considered too complex (Siau et al., 2001), Somehow it is less oriented toward practical use and how it could be implemented in more practices and domains (Henderson-Sellers, 2005).

2.3. Design patterns

Design patterns have been defined as common solutions for some of the most recurring software problems in software development projects (Gamma et al., 1995). One of the main goals of using design patterns is to facilitate informal communication between software engineers and developers to make the software development process more efficient. The second most important use of design patterns is in the software development implementation phase. There is a misconception about design patterns in the implementation phase of software design. Many believe design patterns are disconnected from the implementation phase and code in software development, while in fact, they are truly connected to the code and code generation process (Avgeriou & Zdun, 2005; Riehle, 2011; Riehle & Züllighoven, 1996).

To aid documentation of the software development process is the third main use of design patterns. For example, a software engineer might use objects and classes to describe the structure of a software system. He or she may

use a set of design patterns to show how the software system works using pattern elements, structure, and relationships. Design patterns provide a specific vocabulary for software engineers and developers to efficiently document the software development process (Riehle & Züllighoven, 1996).
It should be mentioned that all definitions and classifications related to design patterns in this thesis are based on the "Gang of Four (GoF)" book on design patterns. This thesis will introduce design patterns and their classification based on GoF's definition and show how they can be integrated into the model-driven architecture (MDA) approach for developing geospatial web applications. In the following two chapters, one design pattern will be selected for MDA integration to develop a geospatial web application. Selecting and implementing patterns is based on the feasibility and experience needed to use them for developing an application. Finally, matters regarding the usefulness of the selected pattern in the development process are outside of the scope of this research. There will be no discussion on the pattern's positive or negative influence and the pattern selection criteria. These subjects are way beyond this research and should be examined in another research.

2.3.1. A short history on software design patterns

The notion of patterns and pattern language was originally derived from the work of Christopher Alexander. Alexander, a famous architect in the twentieth century, was looking for ways to improve the architectural design of buildings and came with the idea of using "patterns" for better communication between building users and architects. According to Alexander, there was an essential missing piece in the twentieth century's architecture (G. Rode, 2008). He believed that inhabitants and users of the buildings should be more involved in the building design process. This way, the final structure would be more habitable (physically and spiritually). Therefore, he suggested using a set of design patterns that are easily understandable for both users and architects in the design process to ensure communication and knowledge transition between different stakeholders (e.g., users, owners, architects). Furthermore, he emphasized two critical issues that each pattern should address: first, the notion of a design problem presented in a certain level of abstraction of that real-life problem, and secondly, the solution for that specific problem (G. Rode, 2008). This way, each specific pattern would be suitable to use for a particular problem.

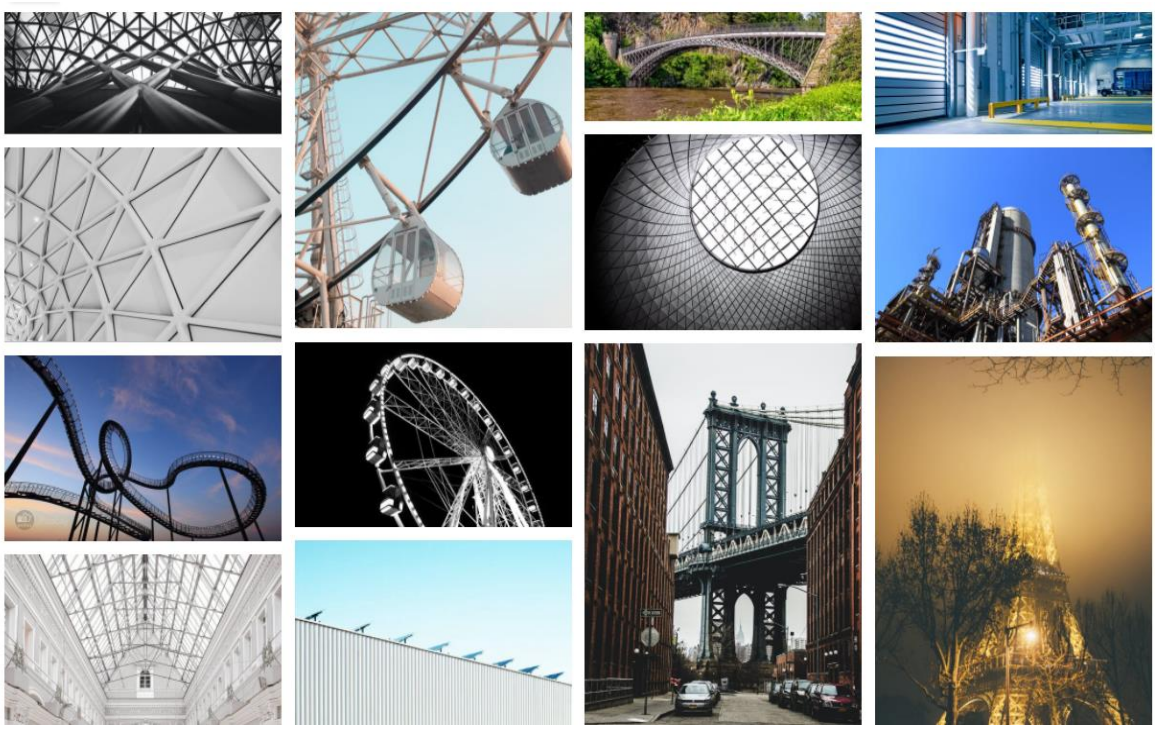


Figure 2-3- Various design patterns in architecture

Since then, there have been promising results in using patterns in other scientific fields, especially in the computer science domain known as software design patterns. In 1991, Beck and Cunnighum applied Alexander's ideas on design patterns in computer science. They created a pattern language with five major pattern categories (G. Rode, 2008; J. Rode et al., 2005). In addition, Jim Coplien 1991 published a book about C++ idioms and their applications. While he did not specifically mention design patterns in his book, his work significantly influenced the future works on design patterns (Coplien, 1991; G. Rode, 2008).
In the early 1990s, several people worked on design patterns and discussed their use in computer science. However, the term software "design patterns" became popular after a book published by Gamma et al. in 1995. The authors of the book became known as Gang of Four (GoF), and the patterns represented in their book is known as the Gang of Four design patterns (Fowler, 2006; Gamma et al., 1995; Riehle, 2011; Riehle & Züllighoven, 1996; G. Rode, 2008).
Since then, there have been several books and publications (scientific and professional) about design patterns and their use in software development. However, the GoF design patterns are still the most popular pattern classification which several works and publications have been created based on them (Fowler, 2006). They are still the prevalent pattern classification both in the scientific community and among software engineers and developers.

2.3.2. GoF design patterns

The Gang of Four design patterns classification contains twenty-three design patterns. This section of the thesis will review GoF design pattern classification basics with an overview of the patterns. Also, some of the more helpful design patterns will be reviewed more specifically. The pattern format introduced in the "design patterns" book by Gamma et al. (1995) has been named Gang of Four (GoF format) since then in the software industry and scientific community.

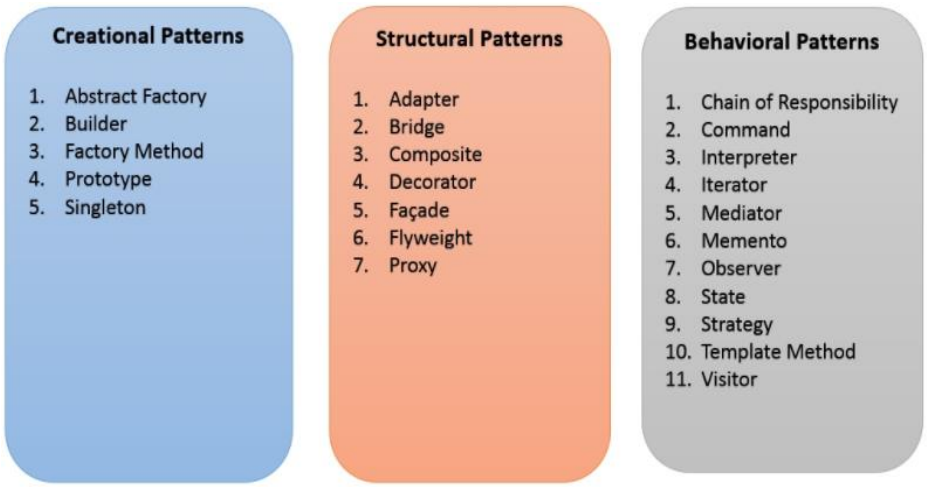


Figure 2-4- GoF design pattern classification

This format is the most popular format for pattern classification in the field. Many have commonly used it as the basis in several software projects and practical works, and other works on design patterns and pattern classifications (Fowler, 2006; Riehle & Züllighoven, 1996). For the rest of this thesis, we mean those twenty-three patterns identified by GoF whenever we are dealing with software design patterns.
As mentioned in the previous sections, the GoF categorizes software design patterns into twenty-three patterns. It describes the structure and concepts of each pattern in detail (explaining almost all classes and object types involved in each pattern definition, the abstraction level that pattern provides for the object-oriented software system). The main goal is to help software engineers and developers solve a specific design problem within a particular context (Gamma et al., 1995).

The pattern classification in the GoF format is based on two criteria: the pattern scope (the focus of the pattern, which could be on objects or classes) and its primary purpose, which could be either creational, structural, or behavioral (Gamma et al., 1995). The pattern scope describes whether the pattern applies to objects or classes. Class patterns usually deal with the classes and the relationships and connections between them, while in the object patterns, the emphasis is on objects, connections, and collaboration. Regardless of their scope, almost all object patterns use the Object-oriented inheritance notion and usually are more dynamic than class patterns.
The purpose or the main goal of each pattern refers to applying that pattern to solve a certain kind of problem. For example, in creational design patterns, the main goal is the process of object instantiation. With structural patterns, the focus is on objects and classes and how these entities could be used to form a more extensive and more integrated software structure. Furthermore, in behavioral patterns, the focus is on defining algorithms and rules to govern the responsibilities of each object and class and manage their interrelationships.

2.3.3. The use of design patterns

Like any other tools and methodologies, design patterns should be appropriately used concerning the problem context and other important considerations (e.g., based on each project's circumstances in the software design process). If not used correctly and within the right circumstances, it might lead to not results that are not good enough. It might decrease efficiency and may add more complexity to the project or even become inconsistent with the principles of object-oriented software development (G. Rode, 2008).
Using design patterns in the development process does not guarantee a better software design (Gamma et al., 1995; Vlissides, 1997) and might result in more complexity and code duplication. There might be several dangers in misusing design patterns, even when someone tries to follow the principles of OO software development without recognizing the need and urgency for it (Fowler, 2006). The most crucial point to notice when using design patterns in the software development process is to recognize that they are not just simple recipes containing some programming rules and tricks (Livshits, 2005).
Using design patterns, one should acknowledge other vital aspects of the project, such as the development context and problem environment that each design pattern is suitable for (Vlissides, 1997). When using design patterns, it is essential always to consider the basic principles of object-oriented software development such as reuse, maintenance, and modification and not to use design patterns with the price of ignoring those main principles (Wirfs-Brock et al., 1990).
This research aims to show how some design patterns could be integrated into the Model Driven Architecture (MDA) methodology for geospatial web application development. So, while some of the considerations and drawbacks of using design patterns in the development process are mentioned, they will not be applied for selecting and implementing design patterns.

2.4. Web applications

This section discusses web applications, their definition, their development history, and the methods used for developing them. After a quick review of these concepts, the geospatial web applications as the specific web applications will be explored, including their definition, origins, and development trends.
A web application is a piece of computer software that runs on a web browser over the internet. It incorporates functionalities beyond a set of interconnected web pages and navigation links (Jazayeri, 2007). A web application remotely initiates all the data analysis and information processing functions (and components) from the client system (e.g., browser) and runs them (partly) on a web server (Bruno et al., 2005; Finkelstein et al., 2001).
We can consider a web application as a software system that consists of one more data source, a set of database functionalities (back-end), a mechanism to interact with data (front-end), with communications and user interaction are taking place over a network (Internet) all controlled through the user's browser (Y. F. Li et al., 2014). The functionality of a web application depends on specific user needs and requirements, actions, and inputs

(Jazayeri, 2007; Molina-Ríos & Pedreira-Souto, 2020). The new generation of web applications support a diverse range of users with their specific needs, from individuals and small businesses with few users and limited use to national and international firms and big companies with hundreds of millions of users.

- There are few general characteristics in most web applications (Jazayeri, 2007):
- Constantly changing needs and requirements
- The high number of end-users
- Various stakeholders
- Complex content
- Typically, short development lifecycle.

Here are the main differences between conventional software and web applications (Bruno et al., 2005; Deshpande et al., 2003; Deshpande & Hansen, 2001; Jazayeri, 2007; Lowe, 2003):

- The development period of web applications is extremely short comparing to the conventional software development
- Web applications need constant development and re-versioning while it is happening once a while for conventional software
- Web application development requires a much smaller development team
- Web applications are extremely sensitive to new technologies and methods
- There is a lack of testing process in web applications, while this is an important part of the conventional software development process
- There are more security threats for web applications than conventional software
- Web applications constantly have to deal with rapid evolution in development technologies
- Users can run web applications from almost any type of computer (e.g., mobile phones, tablets, and PCs) regardless of its specifications.
- Web users need to be connected to the internet.
- Using web applications, there is no need for space to store data or software

2.4.1. History, current situation, and future trends in web application development

The World Wide Web (WWW) has been around since the early 1990s. The initial goal of the Web was to provide a simple and reliable way to access information for computer scientists and research community. Based on this goal, the first web applications were designed simply for accessing and search information within the web. These applications were static applications for just displaying texts and images through some interlinked web pages. There has been a significant change since then. In the last few decades, there has been a significant improvement in the quality and quantity of web applications, their use, and the number of users (Bruno et al., 2005; Y. F. Li et al., 2014). From being just a repository of HTML pages to provide access to other static web pages with information (primarily scientific!) to highly powerful platforms in different types of client computers (e.g., PCs, laptops, smartphones, tablets) that provide various services for nearly all aspects of our daily life. Nowadays, using new technologies, platforms, programming languages, and their respected platforms, creating dynamic web applications with new data transfer models and user coordination and collaboration is doable. The current breed of web applications (the new generation) allows users to store, process, and share information. They also improve communication and collaboration between users. Figure 2-5 shows the evolution of the web and related technologies and introduces main trends in each development era.

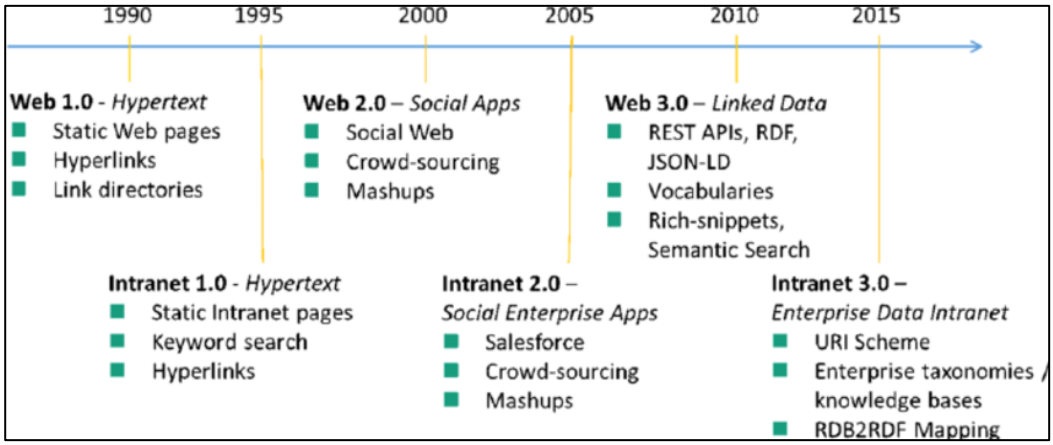


Figure 2-5- Evolution of web, its related technologies and trends

2.4.2. Characteristics

Today web application should exhibit the following characteristics (Schwinger & Koch, 2006; Suh, 2005; Xiao-wei & Xue, 2011):

- They have to loosely coupled with data and development technologies and platforms, because of rapid pace of technology development and constant changes in data sources.
- They should have the ability to perform well even in times of peak usage and high demands. Since lots of services and businesses are dependent on them, a short interruption could lead to lots of problems.
- They have to be secure and free from network security risks and hacks. Threats to web application security are a reality and happening across the globe. Common security vulnerabilities of today's applications are broken authentication, sensitive data exposure, broken access control, security misconfigurations, insecure deserialization, and insufficient logging and monitoring.
- They should be well documented which makes sharing the development process with the new team members, updating, and maintaining application much easier.
- Accessible (for different types of users with and without limitations)
- Interoperable (the ability to operate on different platforms, browsers, and with different technologies)
- Compatible with new changes (new technologies and development platforms)
- Scalable
- Maintainable (easy to update and add new functionalities, easy to debug and test)
- Robust and reliable
- Functionally complete and correct

2.4.3. Geospatial web applications

The internet provides access to almost unlimited processing power, geospatial data, and information, regardless of the user's location and other barriers such as installing a GIS software, having a valid license, and other limitations such as specific computer hardware. Nowadays, geospatial web applications are making a significant improvement in the way we can acquire, analyze, store, share and disseminate geospatial data (Adnan et al., 2010; Alesheikh et al., 2002; Neumann, 2008).

Since there is no exact definition of the term "geospatial web application" in the literature, in the following sections, some definitions and explanations about the main concepts and key terms in this domain will be provided.

When we are talking about geospatial data on the web, several terms come to mind. Web GIS, internet GIS, web mapping, GeoWeb, mashups, geospatial web applications, and etc. (Neumann, 2008). Most of the definitions and explanations of the above concepts are the same as defined by several researchers, though there are slight

differences in some cases. Therefore, it is good to clear things up first. The idea is that after reviewing these provided definitions, one could differentiate between the geospatial web applications and all the other related concepts in the field.

2.4.3.1. Web mapping

Many researchers and GIS professionals define web mapping as a set of procedures for gathering and processing geospatial data and visualizing this data using maps through the world wide web (S. Li, Dragicevic, & Veenendaal, 2011; Neumann, 2012; Techopedia, 2021; Veenendaal et al., 2017; Wikipedia, 2021). In this definition, the emphasis is on the Web as the primary environment for providing GIS functionalities.
The Open Geospatial Consortium (OGC), a de-facto standardization body responsible for defining widely adopted interfaces between geospatial data and information and the web, defines the term Web mapping as "a dynamic query, access, processing, combination and portrayal of different types of spatial information over the Web" (Open Geospatial Consortium (OGC), 2021). There are four main components in most of these definitions: geospatial data, software for data processing and geospatial analysis, data visualization in map format, and the World Wide Web (Kemp, 2008; Veenendaal et al., 2017).

2.4.3.2. Web GIS, Internet GIS

Web GIS is a geospatial application that uses web technologies to obtain data and information, processing this data, and disseminate it. It also uses the web for communicating between these three main components (Veenendaal et al., 2017).
Internet GIS is a term that, in some cases interchangeably used with Web GIS. However, one can argue that internet GIS has a broader vision with respect to the technology applied and allow us to have access and interaction with a tremendous amount of information in different formats (e.g., texts, graphics, sound, and software) over the internet. However, Web GIS is the most commonly used term for online GIS (Fu & Sun, 2010).
The main difference between web mapping and web GIS is that the web mapping concept generally focuses more on mapping and providing geospatial functionalities for different applications and users on the web and within the web context. In contrast, in web GIS, other components of a GIS system, such as acquiring data and processing it using the web, are crucial as mapping functionalities (Kuria et al., 2019; Veenendaal et al., 2017).

2.4.3.3. Mashups

Mashup is "a piece of music created by digitally overlaying an instrumental track with a vocal track from a different recording" or it could be "a Web service or application that integrates data and functionalities from various online sources" (*Mash-up | Definition of Mash-up by Merriam-Webster*, n.d.). The term map mashup refers to a website or application that mixes different types of information (geospatial and non-geospatial), software (geospatial processing tools), and web services and maps to provide a unique and integrated service with a single view (Batty et al., 2010).

2.4.3.4. Definition

Although there is no distinct definition of the term geospatial web application in the literature, it can be inferred that the term roughly equals the term "GIS web applications" in some articles and is based on the prior explanations. Looking at the problem in this way, it seems that the definition provided by Kuria et al. (Kuria et al., 2019) best suits this concept. A geospatial web application should have the following five characteristics:

1. It should provide unique geospatial services and solutions for a range of issues in the scientific, business, and social domains.

2. For the user interface and visualization, it should use digitally based maps
3. It should allow users to overlay spatial layers over base maps
4. It should contain one or more geodatabases (Hojati, 2014) and provide the possibility to access third-party data sources and information. The users should be able to obtain, create, manipulate, and remove these data through the application
5. It should provide some geospatial information using web GIS tools (for data processing, analysis, and visualization) and other third-party web services.
6. It should be able to integrate geospatial data with non-geospatial data
7. And finally, the application's primary purpose should be geospatial. For example, while an online shopping app might use apps and some GIS techniques (for the product delivery), we cannot consider it a geospatial application because its primary goal is shopping.

2.4.3.5. Benefits

Thanks to the recent developments in information networks, the exponential increase in computing power, and the incredible developments in smartphones, tablets, and other location-aware devices, there has been an unprecedented growth in the number of geospatial web applications. Geospatial web application help to overcome some of the limitations with the desktop GIS, such as:

- High costs of creating a desktop GIS for developers and high costs of buying software for end-users
- Difficulties of getting updates and reaching information
- The constant need to update geospatial data and GIS software
- The high cost and much effort needed to find the required geospatial data (Adnan et al., 2010; Alesheikh et al., 2002; S. Li, Dragicevic, Bert, et al., 2011)

Compared to desktop GIS, geospatial web applications are cheap and easy to develop. They are highly suitable for acquiring real-time data through a network, and there is no need to update data and software constantly. There is a higher level of accessibility to geospatial data in these applications, and they have the built-in capacity to disseminate geospatial information on the network (Gordillo et al., 1999; Veenendaal et al., 2017).

2.4.3.6. History of geospatial web applications

The early geospatial web applications were tools allowing individuals and organizations to publish their maps on the internet. There were only a few limited options to do some basic operations (such as zooming, panning, and layer change) available for the users. There were static, and there was not any user interaction possible (Neumann, 2008).

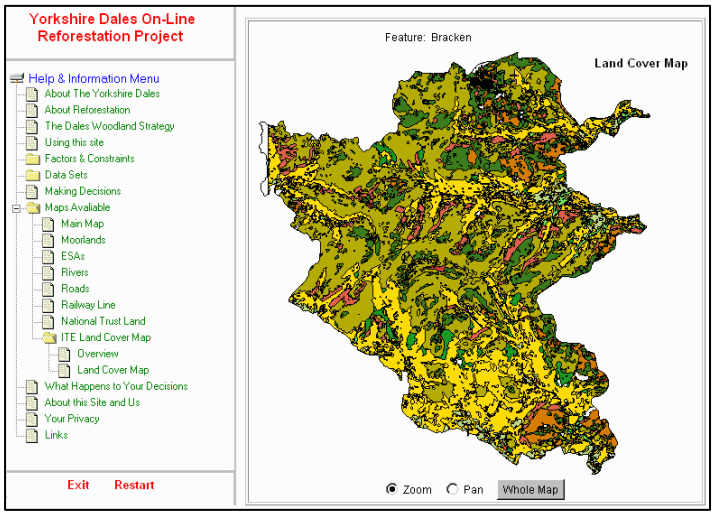


Figure 2-6- Example of an early web GIS (photo from http://www.geog.leeds.ac.uk/papers/99-1/gc104bd2.gif)

There have been several noticeable improvements in that area since then. For example, today's geospatial web applications provide much more geospatial functionalities (more than a simple view) for users along with many other capacities of desktop GIS (e.g., geospatial data processing functionalities) and new possibilities beyond desktop GIS such as new possibilities to share data and processing power, the notion of collaborative GIS, and access to real-time data using GPS and different sensors (Veenendaal et al., 2017).

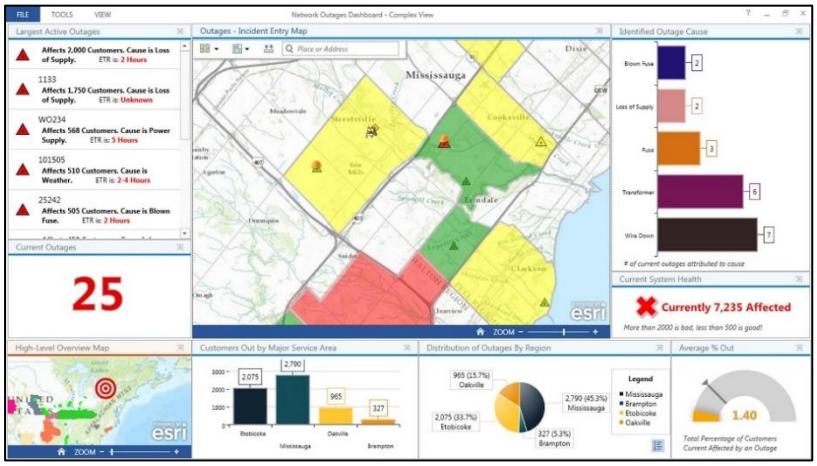


Figure 2-7- A modern GIS dashboard (esri.com)

2.4.3.7. Criteria for developing geospatial web applications

While in recent years, there have been significant improvements in the quality and increase in the number of geospatial web applications (thanks to the technological advances in computer software and hardware). These improvements have resulted in incredible changes in use, users, and the quality of services provided by geospatial web applications. However, they are still struggling with several new challenges. Some of the challenges ahead are:

- Linking geospatial data and information to other non-geospatial data
- Network and processing limitations to analyze and handle big geodata.
- The massive amount of geospatial data from different sources
- Location intelligence and using AI in geo

- and security issues
- Data privacy (e.g., regulating access to public geodata)

To solve some of these challenges, geospatial web applications should be able to acquire, process, and view huge amounts of geospatial data. They have to be able to work with different datasets and APIs and provide enough processing power to deal with huge geospatial data. They should have the ability to be upgraded based on the data and technology in use. One way to achieve these goals is to find and use a suitable web application development methodology.

2.5. Web application development

In the past few decades, software development has experienced exponential growth due to the rapid increase of software companies globally and the current trend of using the latest technologies in software products. At the same time, several web development methodologies have been introduced to replace the more traditional ones. During the development process, software engineers and web developers face several challenges related to the web application life cycle, implementing new features into the application, and constant technological upgrades and changes. As a result, new development methodologies must be created to answer all these new changes and requirements (Molina-Ríos & Pedreira-Souto, 2020).

There are two different approaches toward web application development: traditional development methodologies and OO development methodologies. Since there will be a section devoted to the web application development methodologies, here we will discuss traditional and agile development methodologies for developing web applications.

In traditional software development, customers usually are not fully aware of the application's requirements. Also, software engineers and developers are expected to incorporate some new functionalities into the application to satisfy customer (end-user) needs. Therefore, the whole development process is strictly defined and planned, and usually, there is not so much room for sudden changes or adding new features to the application (Chan & Thong, 2009). Therefore, many traditional web development methods are considered inadequate for dealing with the challenges of development in current web applications (Standing, 2002), and developers tend to use agile methodologies more and more.

On the other hand, agile development methodologies emphasize development teams to find the best methodology for each development project and provide some elasticity and ease for professionals working within development teams. Agile practices, which resulted in significant improvements within the software development field, involve discovering requirements and developing solutions through the collaborative effort of self-organizing and cross-functional teams and their customers (Chan & Thong, 2009).

2.5.1. Web application development methodologies

The internet and web and constantly evolving technologies have had a significant impact on the business around the world. Along with the increasing number of web applications (today, many businesses use the web and web applications somehow to conduct their business), there has been a rapid increase in software development companies and individual developers. While they are using different approaches and methodologies to develop their products, there are still some shortcomings and challenges with their methodology. They are constantly looking for new methodologies and approaches to create better applications (Molina-Ríos & Pedreira-Souto, 2020). That is why using a suitable development methodology for a web project matters so much. With several methodologies for developing a web application, specific requirements of each application, complexity issues, and time limitations should be considered when selecting the most suitable methodology (Molina-Ríos & Pedreira-Souto, 2020; Schwinger & Koch, 2006).

Methodologies for web application development either rely on traditional software development patterns (like ER entity-relationship) or promote themselves on the OO modeling paradigm. They are mainly developed based on the UML.

Web application development methodologies could be classified into five main categories. The first category includes data-driven methodologies. These methodologies originated from the database systems field, basically are using ER (entity-relationship) modeling concept with some enhancement (Brambilla et al., 2008; Ceri et al., 2004). Their primary goal is to model those web applications with a data-driven structure. Some of the prominent examples of this modeling category are hera (Charatan & Safieddine, 2002), Web Modeling Language – WebML (Schwinger & Koch, 2006), and Relationship Management Methodology -RMM (Standing, 2002).

The second category covers hypertext-oriented methodologies. Hypertext-oriented methodologies are based on hypertext systems (Garzotto et al., 1995) and are developed and enhanced to model the hypertext dimension of web applications (Baresi et al., 2001). The most notable modeling methodologies based on this paradigm are Hypertext Design Model - HDM (Plessers et al., 2005) and its extension - W2000 (Charatan & Safieddine, 2002), Web Site Design method – WSDM (Koch et al., 2002), and HDM-lite (Schwabe & Rossi, 1995).

Object-oriented methodologies are within the third category. These web application modeling methodologies are based on either the principles of the OMT (object modeling technique) or UML. In many cases, UML is the preferred modeling language to use along with these methodologies. Methodologies such as UML-based web engineering - UWE (Garrigós et al., 2003, 2010), OO hypermedia Design Method - OOHDM (Conallen, 2003), OO Hypermedia method - OO-H (Wimmer et al., 2007), and Object-oriented Web Solutions - OOWS are in this category.

The fourth category includes software-oriented methodologies. Methodologies in this category look to web application development from the traditional software development perspective and use standard software development techniques. Methodologies such as Web Application Extension - WAE or WAE2 (Rossi et al., 2001) are in this category.

Finally, the fifth category includes model-driven engineering-oriented methodologies. Prominent use of models in this category represents the ultimate trend in the web application modeling field. These methodologies are generally based on OOP notation (Schwinger & Koch, 2006). The new web application development methodologies are always built on top of the existing methodologies. Figure 2-8 describes the chronological occurrence of different web modeling methodologies based on their origins.

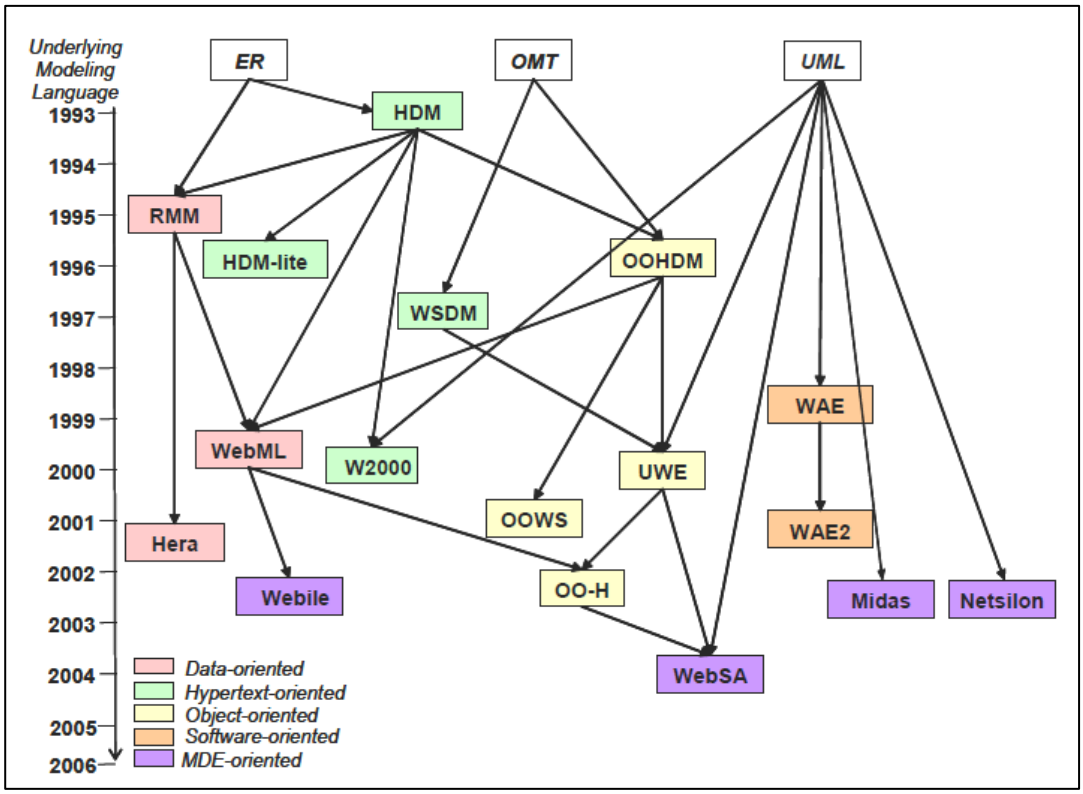


Figure 2-8- The evolution of web development methodologies (Schwinger & Koch, 2006)

HDM-lite, which is an advanced version of HDM, is designed so that it will be able to generate web applications automatically. The other modeling enhancement of HDM, W2000, is trying to model web applications based on their user-centric and hypertext-centric perspectives (Schwinger & Koch, 2006). RMM model is built on top of the ER notation. Hera is another web application modeling approach based on the ER paradigm and uses some RMM notations. WebML, which uses the WebRatio modeling tool for web application modeling and code generation, is one of the most understandable and mature web modeling languages out there.

To model web applications with various navigational access preferences, OOHDM modeling methodology is recommended since this methodology emphasizes the importance of the navigational concept in web applications. In recent years, there has been some modification and enhancement on OODHM methodology to support framework modeling and personalization of web applications (Huang & Chu, 2014).

The focus of WSDM is to understand user requirements using a methodologic approach. On the other hand, UWE's approach is based on UML notations and meta-model consistency checking. OO-H is one of the newest paradigms in web application modeling, which combines many positive aspects of WebML, OOHDM, and UWE. This methodology uses the VisualWADE tool for generating model-driven code in an automatic manner (Knapp et al., 2003; Koch et al., 2002; Schwinger & Koch, 2006).

OOWS is like OO-H and uses the same OO approach. OOWS mainly relies on its notation but sometimes uses UML notations for some parts of the application development. WAEZ is a UML-based approach mainly focused on the application logic and its distribution. Finally, WebSA has been proved itself as a reliable modeling methodology for the architecture of web applications (Koch et al., 2002; Schwinger & Koch, 2006).

Originally developed to build data-intensive web applications, Hera is a model-driven web development methodology focusing on the application's design aspects (Garrigós et al., 2003, 2010). Furthermore, facilitating design development. For this, Hera provides an environment for the collaboration of external parties based on the separation of concerns (Torres et al., 2012).

Modeling Method	Modeling Paradigm	Notation	Evolving	Requirements Modeling	Content Modeling	Hypertext Modeling	Presentation Modeling	Customization Modeling	Structure and Behavior	Process / Approach	Tool Support	Generation	Strengths
HDM-lite	HT	ER + own notation	×	×	×	✓	✓	×	s	own	generation tools	auto	process for model transformation, automatic generation
Hera	DB	ER + RMM+ own notation	✓	×	✓	✓	✓	✓	s + b	own	authoring & generation tool	semi	model-driven development
OO-H	OO	UML + own notation	✓	✓	✓	✓	×	pers	s + b	own	modeling- & generation tool	auto	tool for automatic generation
OOHDM	OO	UML + own notation	✓	✓	✓	✓	✓	pers	s + b	own	×	×	powerful concepts for contextual navigation, personalization
OOWS	OO	UML + own notation	✓	✓	✓	✓	✓	×	s + b	own	modeling- & generation tool	auto	advanced (commercial) tool for automatic generation
RMM	DB	ER + own notation	×	×	✓	✓	✓	×	s	own	authoring tool	semi	hypertext modeling based on ER-model, predefined process
UWE	OO	UML	✓	✓	✓	✓	✓	pers	s + b	RUP	extended UML tool & generation tools	semi	UML-based method, model-driven development, aspect-oriented customization
W2000 (HDM)	HT	UML	✓	✓	×	✓	✓	pers	s	×	extended UML-tool	×	user-centric hypertext modeling
WAE2 (WAE)	SW	UML	✓	✓	✓	×	✓	×	s + b	RUP	standard UML-tools	×	implementation design, architectural design
WebML	DB	ER, UML	✓	✓	✓	✓	×	pers	s + b	own	modeling- & generation tool	auto	well-elaborated notation, database integration, generation
WS DM	HT	own notation	✓	×	✓	✓	×	×	s + b	own	×	×	user-centric approach for analysis

✓	supported
×	not supported

pers	personalization

s	structure modeling
b	behavior modeling

RUP	Rational Unified Process
own	own process model / approach

auto	automatic generation
semi	semi-automatic generation

DB	data-oriented
HT	hypertext-oriented
OO	object-oriented
SW	software-oriented

Figure 2-9- Web development methodologies and their main characteristics (Schwinger & Koch, 2006)

2.5.2. Model driven software development (MDSD)

Many software engineers and web developers agree that models should be a part of any web development project. However, there is no common agreement on the role of models in the development process, how software engineers and developers can achieve that, and who should play a role in the web application modeling process (Schwinger & Koch, 2006).

Today, web development teams either do not use models in their process or use them only for the initialization of the project. As they move forward in the development process, they usually leave models behind, do not reflect project changes, and update them. As a result, there is no use for models for the rest of the projects (Rossi et al.,

2016). In this section, after providing definitions of models, metamodels, and their applications in the computer science field, the notion of model-driven development (MDD) in software development will be discussed.

2.5.2.1. Models

In computer science, models play an essential role as they help software engineers and developers deal with larger and more complex systems by transforming real-world concepts into the proper levels of abstraction (Ahmed, 2013).

Models are representations of real-world events. Models are artifacts that contain entities, metaclasses, functionalities, relationships, and a means for documentation. The computer science field always used models and their benefits, even before their introduction to the domain. While software engineers and developers support the more structured use of models in the software development process, development teams only use models in the initial phase of project development. They usually ignore models and their use during the other phases of software development and don't update them based on the changes in the project (Molina-Ríos & Pedreira-Souto, 2020).

2.5.2.2. Model-driven development

One of the major problems with the current programming languages and using them outside the context of a model is that they are usually too focused on specifying how a software solution should work rather than mention what the solution should be (J. d. S. Saraiva & Silva, 2009).

The model-driven development (sometimes called model-driven engineering or model-driven software engineering) paradigm in software engineering promotes models in all stages of software development. It emphasizes the importance of models and model transformations as the central part of the solution specification (J. Saraiva, 2013). In MDD, basically, all the other requirements in a software project, such as source codes, dependencies, entities, and documentation, can be derived from models (Brambilla et al., 2012; Schmidt, 2006).

Currently, there are several software development approaches based on MDD. However, it is essential to note that MDD does not delineate any approach over the others. As already mentioned in this section, MDD itself is a paradigm in software development that these approaches could address, and it is independent of any programming language of technology (Schwinger & Koch, 2006).

There are two major benefits of using MDD in web application development. One is decoupling different aspects of the application (application architecture, user interface and view, datasets and information structure, business logic) from another. The other is creating a distinction between platform-specific issues and the rest of the application. This way, the application model and its logic could be used regardless of the development platform (Meservy & Fenstermacher, 2005; Taleb et al., 2007).

2.5.2.3. Model-driven web engineering (MDWE)

Model-driven web engineering is a major field in web engineering based on using model-driven methodologies for web application development (Rossi et al., 2016). Therefore, MDWE (same as its parent discipline MDSE) tries to develop web applications mainly by developing conceptual models and transformations to derive platform-specific models from them and finally generate code.

Four main reasons make it necessary to implement models in web engineering. The first reason is the rapid improvements in hardware and internet infrastructure. Different types of networking devices and computers (e.g., smartphones and tablets) along with huge improvements in the computational power of these devices, with the vast cover of fast internet in many areas around the world, means more use of the internet and more complex web applications (Brambilla et al., 2012; Moreno et al., 2007; Schwinger & Koch, 2006; Taleb et al., 2007). The second reason is constant technological advancement in web development from programming languages to frameworks and new development methods. The third reason is the vast increase in the market demands for web products.

There is an increasing need for more support and update of web applications. Nowadays, users use the internet and web applications 24/7, and every day there are new applications and functionalities of the web in our lives. Moreover, the final reason is a better understanding of web applications and the internet by web developers and software engineers, which resulted in acknowledging that we need new methodologies to develop web applications (Deshpande et al., 2003; Taleb et al., 2007).

The rapid growth of the web as a platform for software development (with all sorts of applications in our daily lives) in recent years made a noticeable impact on the software engineering discipline. Web engineering is looking for ways to solve some of the problems and challenges in web development. By applying a set of integrated, systematic, and quantifiable software development methodologies to develop, deploy, test, and maintain web applications (Schwinger & Koch, 2006).

2.5.2.4. Web modeling tools

As has been explained in the previous section, all the web modeling methodologies represent a set of modeling concepts and elements suitable for modeling the specific characteristics of web applications. In addition, they define an application development process, and there are a set of tools and software to support this process. These tools provide a big help for software engineers and architects to (semi) automatically implement the development process and finally generate required models and code based on each methodology (Schwinger & Koch, 2006).

visualWADE

Developed by The Web Engineering Group of the University of Alicante, the visualWADE is a software tool and a set of methods to model web applications. VisualWADE is language independent and based on OO-H methodology. This tool can automatically model web applications and generate codes for them in PHP, JSP, ASP, and XML (Knapp et al., 2003).

VisualWADE integrates a UML model with two views: a "navigation view" and a "presentation view." The presentation view handles appearance, user interface, and application behavior. The visualWADE builds navigation view by creating a class diagram with hypermedia navigation features. A set of interconnected template structures expressed in XML are responsible for modeling presentation view

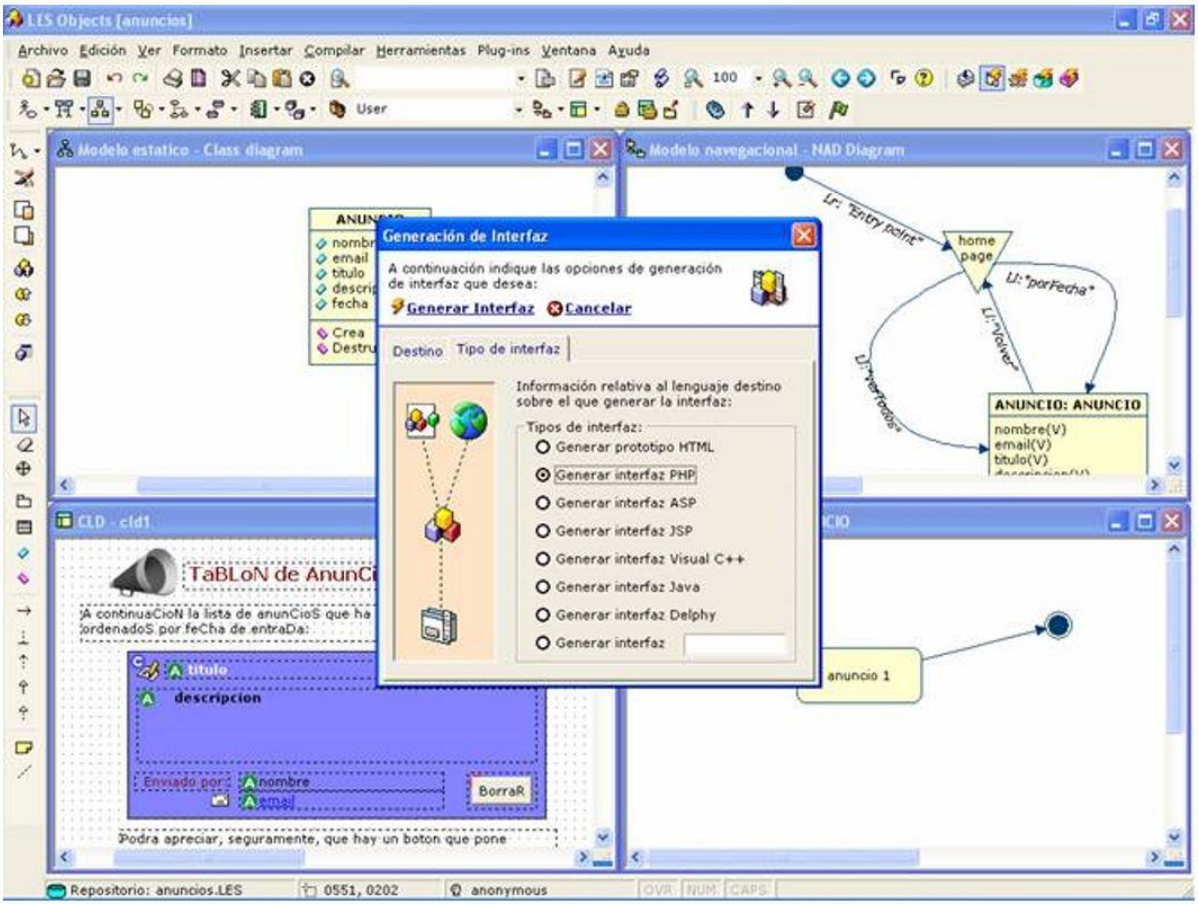


Figure 2-10- snapshot of the visualWADE tool (http://gplsi.dlsi.ua.es/iwad/ooh_project/cawetool.htm)

OpenUWE

OpenUWE is a web application modeling environment based on the UWE modeling framework. This tool is developed by the web engineering group of the ML Munich university. One of the essential features of this tool is that it is developed based on verified standards supported by open source and commercial tools (Ziegeler & Langham, 2002).

openUWE includes ArgUWE and UWEXML. The argUWE is a case tool that could be used for the model-driven development of web applications (Schwinger & Koch, 2006). The UWEXML framework consists of different models and tools for consistency checking, layout editing, and code generation (Sparx Systems, 2019).

ArgoUML

The ArgoUML case tool is the main foundation of ArgoUWE. The agroUWE can support the UWE modeling notion and check model consistency based on a set of OCL considerations (Martínez-García et al., 2015; Wimmer et al., 2007).

Enterprise Architect (EA)

Sparx System's Enterprise Architect (EA) is a modeling tool for designing and developing software systems and web applications. EA is based on UML specification and is suitable for creating business process models and visualizing development processes in software projects and web applications. EA covers almost all aspects of a software development project, from initial phases like system design and basic modeling to more advanced phases such as product testing, deployment, and maintenance.

EA supports code generation for many languages such as C, Java, JavaScript (Schwinger & Koch, 2006). One of the main advantages of EA is its scalability which makes it a useful tool for applications with a high number of users or very complex and advanced architecture (Brambilla et al., 2012; Martínez-García et al., 2015).

Eclipse Modeling Framework (EMF)

Although there are several tools for model-based software engineering and web application development, Eclipse Modelling Framework (EMF) has been one of the prominent modeling frameworks globally. It is well known in the software development community. EMF contains several software modeling tools. Each of them provides a special opportunity for a specific software modeling task, making the toolset interesting for many developers with different software and web projects.

EMF allows developers to define metamodels based on Ecore modeling language, which is the heart of EMF. EMF then uses generator components for metamodel production, model manipulation, and edition. This way provides a fantastic tool to develop various kinds of models for a set of different software and web applications (Brambilla et al., 2008).

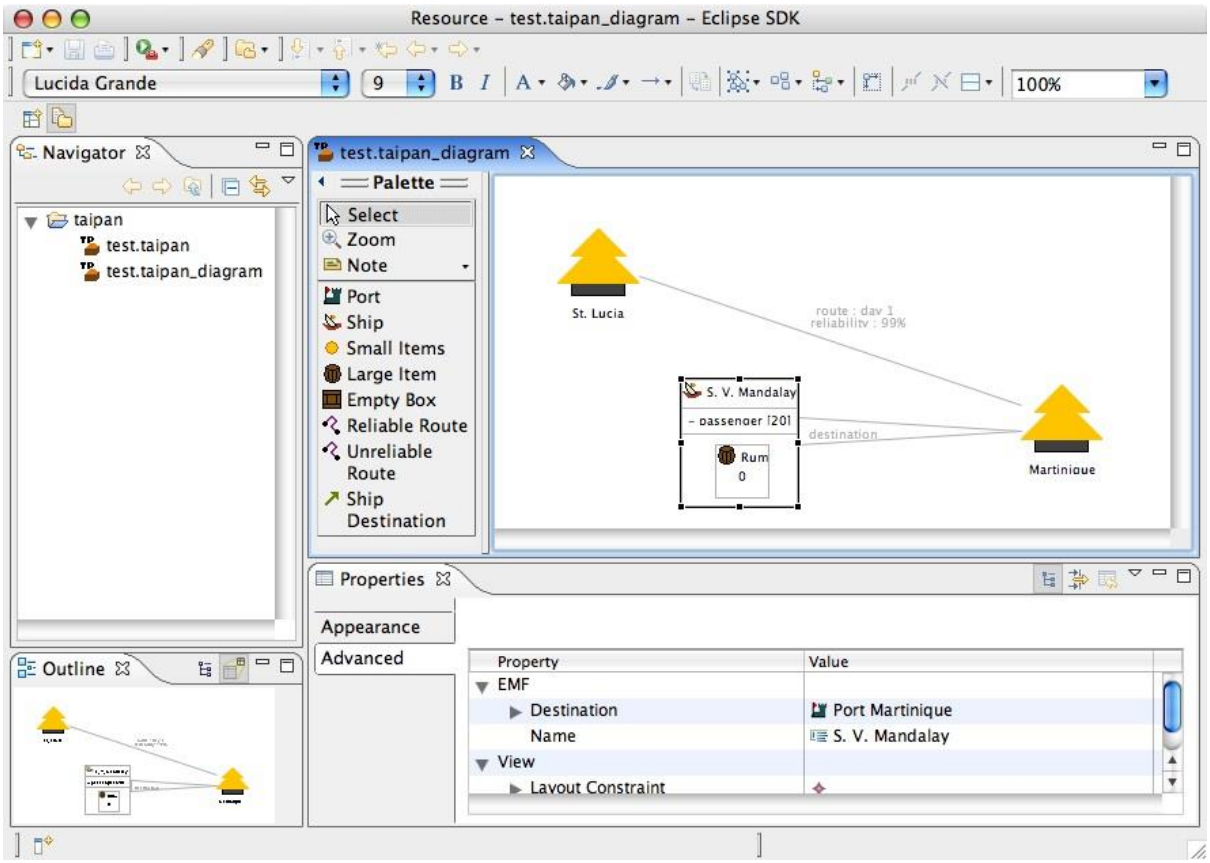


Figure 2-11- Eclipse modeling framework interface (https://wiki.eclipse.org/File:Taipan_diagram.png)

WebRatio development framework

WebRatio is a low-code platform that enables us to develop web applications faster and more efficiently. Based on web modeling language (WebML), the WebRatio application development tool is a model-driven platform to create web applications (Houben et al., 2003).

The code generator integrated into this tool uses XSL to transform models from XML format into the required web component representations or database connections with different formats (such as HTML, PDF, Microsoft Word, and WML). Using the easyStyle tool, WebRatio generates presentations for web pages and automatically transforms them into XSL format. The application architecture in this development platform is based on the MVC-2 pattern. The produced web application will be deployed into a Java runtime framework using a collection of Java components (Schwinger & Koch, 2006).

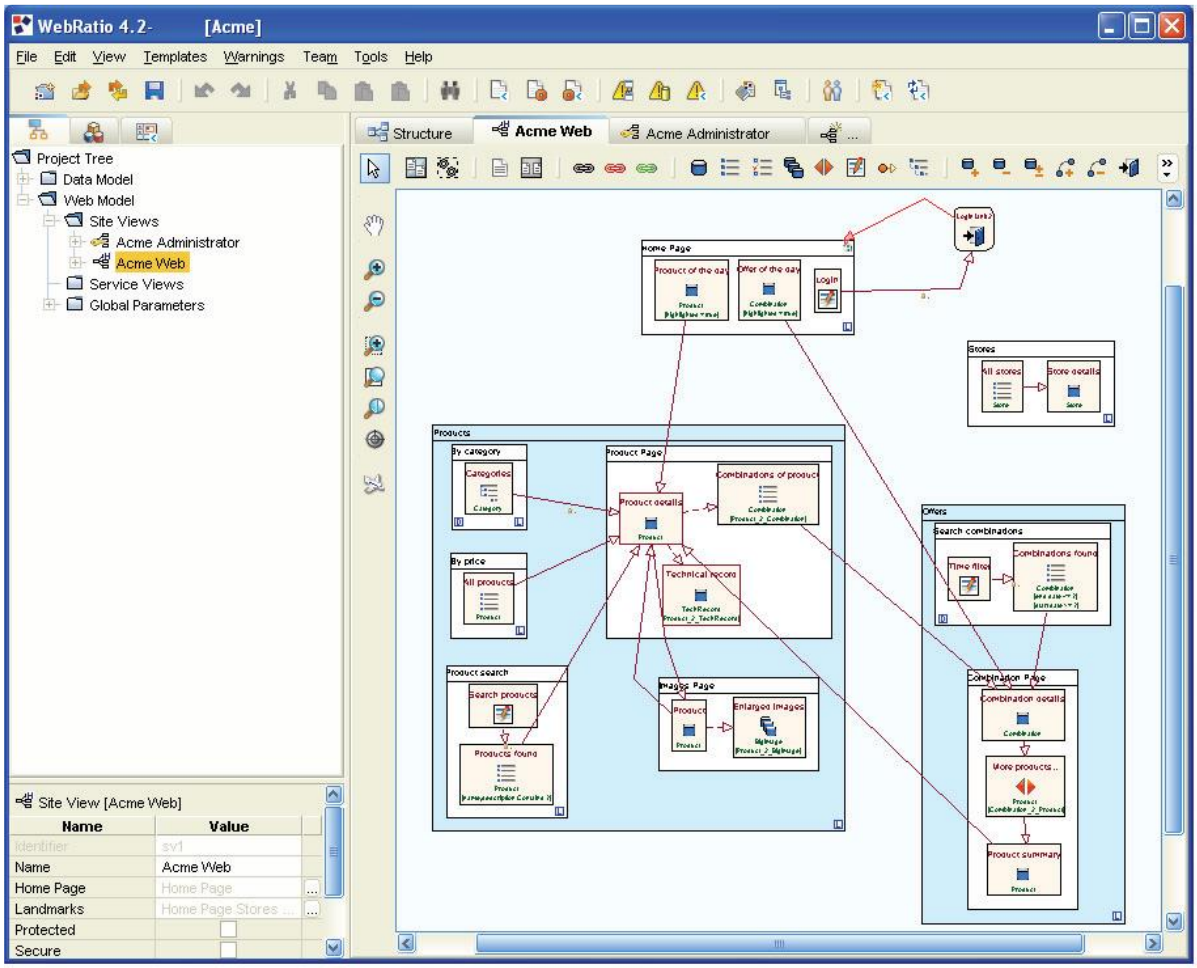


Figure 2-12- WebRatio development environment

2.5.3. Web application methodologies conclusion

Web development projects often contain specific needs and features (i.e., the need to use specific web services, add a new page for the application, or even satisfy a particular user group). So, any of the web development methodologies explained above should have a certain level of flexibility, adaptability, inclusion, and documentation. While each web development methodology has some specific characteristics to address web development issues at some level, they cannot be considered the most appropriate methodology for every web development project. It might be the case that some development methodologies are more suitable for specific phases in application development. At the same time, they could not deliver the expected results for the other parts of the development phases. The choice of web development methodology should be based on the specific needs and requirements of the project, with consideration of future changes and challenges.

The review of methodologies and development tools for web application development provides the knowledge necessary to understand them (at some level) better and to choose the suitable methodology and development tool based on the project specification. Looking at web development technologies reviewed in this section, one can say that OOHDM is a web application development methodology mentioned by several authors as one of the sophisticated methodologies. However, it is not widely used in the industry. Many professionals and companies prefer to use hybrid methodologies since they have some experience using and developing them (Chernichkin & Nikiforova, 2009).

While software engineers and project managers are aware of these methodologies and the possible benefits of using them, personal experience and knowledge acquired during previous projects are among the main factors that impact choosing web development methodology.

However, software engineers and project managers must always consider the web application's life cycle, the conceptual design, the presentation specifications, and the level of each project's complexity before favoring any specific development methodology over others. In other words, the choice of application development methodology should address specific issues and goals of each project, namely project requirements, stakeholder inclusion, the analysis of project functionalities with suitable diagrams, best use of UML devices, prototyping, and user interface (Molina-Ríos & Pedreira-Souto, 2020).

3. RESEARCH METHODOLOGY

3.1. Introduction

The most important part of this research is developing a methodology for pattern-based model-driven architecture (PBMDA) for integrating software design patterns into the MDA process. These two software development approaches have been used separately in software and web application development projects (Buschmann et al., 1996; Gordillo et al., 1999; McArthur, 2008; Prakash & Karri, 2018; Riehle, 2011). However, there are few examples of their integration (D. K. Kim et al., 2017; Seffah, 2015).

This chapter will explore the scientific literature on the MDA and design patterns for developing the research methodology to integrate software design patterns into the MDA. After a thorough analysis of the existing methodologies for integrating software design patterns into the MDA, the best integration method in terms of suitability for developing geospatial web applications and its applicability will be selected and developed further.

At first, in the next section, the MDA approach to software development will be introduced. There will be in-depth explanations about the principles of MDA, MDA's basic models, and other concepts such as metamodeling and DSL. In the end, some of MDA applications will be presented along with the issues and challenges in software development using MDA.

The final section of this chapter is devoted to describing the PBMDA methodology, which is mainly based on the MDA principles. PBMDA tries to integrate the concept of software design patterns into the MDA by using the Pattern Based Model Transformation (PBMT) in the MDA process.

According to the PBMDA methodology, we generated the PIM metamodel using manual CIM to PSM transformation and DSL design principles. Then we developed a metamodel for the PSM based on the generated PIM metamodel and the application's technological specifications (programming language, platform of choice, and desired functionalities).

Since PBMT is the most crucial concept in the PBMDA approach, we tried to describe it in more detail in a separate section. In this section. We explained how PBMT could be used to integrate software design patterns into the MDA. The Observer design pattern will be used to show the effectiveness of this methodology and be used in the next chapter for implementation.

3.2. Model-Driven Architecture (MDA)

In the last two decades, a series of methodologies based on use of models and object-oriented programming principles have been proposed for software and web development. Because of the recent advantages and improvements in class-instance relations provided by OOP languages, now many software modeling tools can be more focused on logical issues such as model editing and defining the structure of metamodel (J. Saraiva, 2013). These methods have a common characteristic: the automatic or semi-automatic transformation of a computer model with a high level of abstraction to other models with a lower abstraction level and, finally, code generation. They are referred to with different names as Model-Driven Development (MDD), Model-driven software development (MDSD), model-driven engineering (MDE), and model-driven web engineering (MDWE) (Mellor et al., 2003; Schmidt, 2006; Taleb et al., 2007). Moreover, they are widely acknowledged by the software engineering and software development community (Favre, 2004).

MDA is an architectural framework for software development that was adopted by the object management group (OMG) in 2001 (Taleb et al., 2007) and is the OMG's particular approach to MDD paradigm. In the MDA approach, the model is the central element in the process of software development (Meservy & Fenstermacher,

2005; Taleb et al., 2007), and model implementation and software code generation happen through a set of automatic transformation rules (Schwinger & Koch, 2006). Applying models in MDA structure, domain experts and software engineers can deal with complex issues in large information systems and enhance communication between organizations, experts, developers, hardware, and software (OMG, 2014).

In the MDA approach for software development, there are three types of models at different abstraction levels that could be used to represent different viewpoints. At first, the information system is analyzed and expressed in the Computation Independent Model (CIM). Next CIM has to be transformed into The Platform Independent Model (PIM). The PIM describes the information system and specifies its computational and technical requirements regardless of any particular platform. The Platform Specific Model (PSM) is the model with the lowest level of abstraction than PIM and specifies the system's technical details with respect to a specific platform and technology.

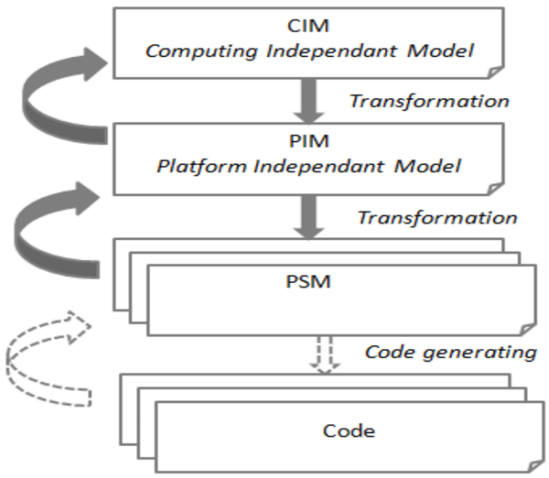


Figure 3-1- MDA basic models (Betari et al., 2018)

The MDA's models and model transformations have to be defined and expressed through a set of modelling languages and standards. MDA is based on OMG's modeling standards including the Unified Modeling Language (UML), the Meta-Object Facility (MOF), the Common Warehouse Metamodel (CWM), and the XMI (XML Metadata Interchange). These modeling standards are being used in almost every development environment (Java, .Net, XML) as a basis for modeling specification and model transformation (Barbosa et al., 2013; Gorton, 2011; Kulkarni & Reddy, 2003; OMG, 2014; Soley & OMG Staff Strategy Group, 2000) in different domains (figure 3-2).

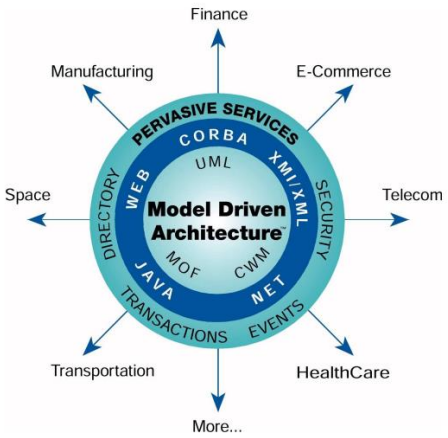


Figure 3-2- The principles of MDA (https://www.omg.org/mda)

UML and CWM as two common modeling languages have been used in several domains. In UML, the focus is on object modeling, and in CWM, the focus is on data modeling. XMI is a mapping standard for model and metamodel (with XML structure) exchange and is based on MOF notation. At the heart of MDA, the MOF is responsible for providing means to define other modeling languages using its four-layered metamodeling hierarchy.

3.2.1. Metamodeling

A metamodel is the structure of objects, entities, and concepts for the definition of a model. Metamodel tries to describe developers' environment (consider a software engineering problem and the environment developers are dealing with that problem) for a particular purpose (solving the software issue). Metamodel uses a set of basic definitions, constructs, rules, and relationships needed to define a model (de Sousa Saraiva & Rodrigues da Silva, 2008; J. Saraiva, 2013). In other words, a metamodel is a language used to define a model. Figure 3-3 illustrates a metamodel and the way it describes a model through a modelling language. Based on these descriptions of metamodel, we could refer to metamodeling as the process of creating a metamodel and generating a model from it (J. Saraiva, 2013).

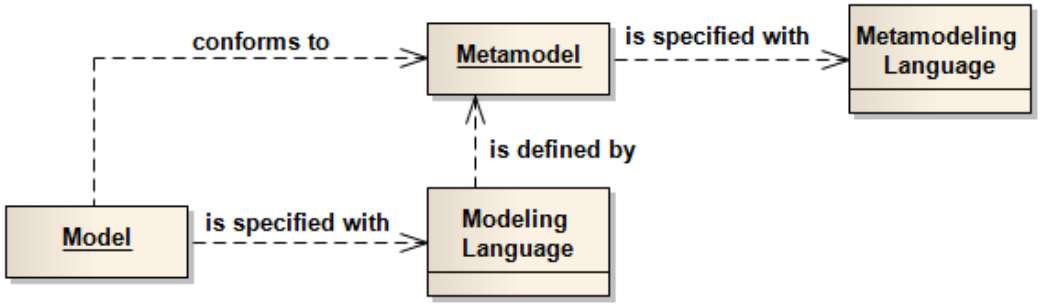


Figure 3-3- The relationship between model, metamodel, and modeling language (J. Saraiva, 2013)

"Talking metamodel" is the best way to evolve, check, and validate the metamodel and to use it to communicate and exercise it with domain experts, developers, and all stakeholders. This makes it possible for domain experts and developers to create and enhance metamodels for better expression and efficient(de Sousa Saraiva & Rodrigues da Silva, 2008) communicate domain concepts and processes.

3.2.2. Meta-Object Facility (MOF)

Along with the UML and CMW, the MOF (OMG, 2011) is one of the main pillars of the OMG's tactic to MDD. OMG has designed MOF so that it can be used as a metamodeling framework to define other modelling langauages in MDA such as, UML and CWM. MOF has this capacity to be used as a metamodel do define itself and other metamodels. This way, both UML CWM and also MOF can be inferred as instances of MOF. Figure 3-4 shows the four-layered structure of MOF metamodels and how it can be used to define a model at different abstraction evels. The left image shows an example of a car modelled using UML and based on these four layered structures (Gorton, 2011; Liu & Wang, 2011; OMG, 2003, 2011, 2014).
Looking at the figure 3-4, we can understand the vital role of the MOF in the MDA process as a meta-meta model, which means it provides a metamodeling language (Gorton, 2011; J. Saraiva, 2013) for models and transformation between them.

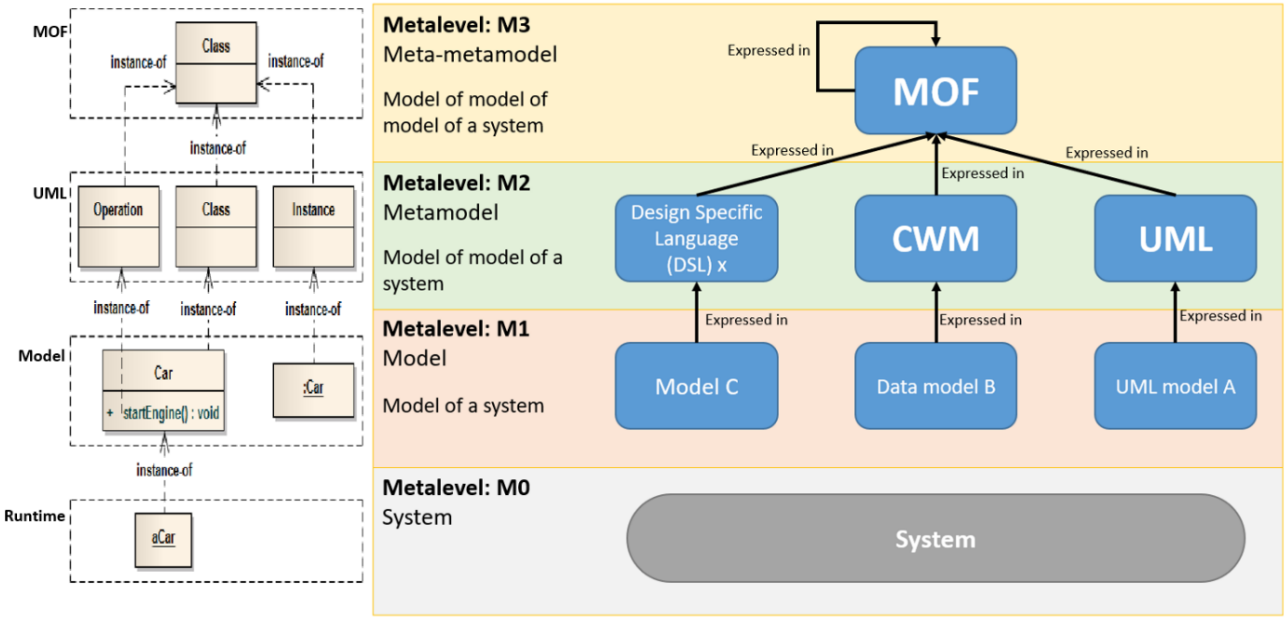


Figure 3-4- the MOF four-layared metamodeling structure (right) - an modeling example in UML (left) - Adopted from (Gorton, 2011; OMG, 2003, 2014; J. Saraiva, 2013)

The modelling language in MDA could be a generic language such as UML which can be used for multiple domains such as IT. The MOF metamodeling structure also facilitates defining Domain Specific Languages (DSLs) which expand MDA applications to other domains beyond IT such as biology, geography (Gorton, 2011).

3.2.3. DSL

While UML is well-suited to use for creating models in several disciplines (e.g., IT) it does not have the required elements to specify particular problems and solutions in some desciplines (e.g., biology, earth science). It is really challenging for experts and developers to for modelling problems and communicate them Using UML. That's where software development introduces DSLs as a set of specifications defined by domain experts and developers to be used where it is not possible to use conventional modelling languages.

DSLs are modeling languages for each specific domain and are used increasingly by domain experts to transform their knowledge to other experts and software developers (J. d. S. Saraiva & Silva, 2009). It is the responsibility of domain experts to capture domain knowledge into a DSL. Application developers can then use the developed DSL to develop and configure the required system (Liu & Wang, 2011).

DSLs are one of the most important concepts in MDA. They are of high value in the modeling process in MDA since they help domain experts and software developers to transform and express domain knowledge into metamodels (Ahmed, 2013).

There are some benefits, using DSLs instead of ordinary modeling languages such as UML (Betari et al., 2018; de Sousa Saraiva & Rodrigues da Silva, 2008; Gorton, 2011; OMG, 2011; J. Saraiva, 2013; J. d. S. Saraiva & Silva, 2009; Thomas, 2004):

- It makes the problem comprehension easier for all stakeholders (clients, domain experts, software developers)
- It is easily transferable between different stakeholders (there is no need to know certain modeling languages)
- It is more flexible and can capture specific aspects of a problem in a domain since it is based on that domain, while modeling languages are more general and usually act better when applied to certain domains
- It is extensible, which means that if we create a DSL for a specific domain, in case of changes to the problem or additional problems, we could easily use the current DSL and extend it (Ahmed, 2013).

By using MOF four layered metamodeling structure MDA provides the opportunity for domain experts and developers to define DSLs for their particular domains. So if it is not possible to express MDA models (CIM, PIM, and PSM) using conventional modelling languages first a DSL (metamodel) should be defined for them and then we can creatre models.

Creating DSLs through metamodeling is considered a significant asset in MDA software development (Ahmed, 2013; Seffah, 2015) that grows over time. Incremental development and consulting domain experts are the mean to come up with a good metamodel (Cáceres et al., 2020; Liu & Wang, 2011; Marcos et al., 2003; Nguyen & Richta, 2014) and upgrade it. Therefore, after a while and several attempts, the resulted DLS could be used in almost any project within that particular domain without any manipulation. This will increase productivity and decrease the amount of time and energy needed for developing any application in this particular domain regardless of its complexity.

3.2.4. CIM

CIM or the application business model describes the application's working logic or its business model. A business model describes abstractly how the business is operating and modeling business processes help to improve communication between software developers, customers, and partners for better controlling the business or establishing an information system (Kherraf et al., 2008; OMG, 2014). CIM has the highest level of abstraction and is the first model in the MDA life cycle. Creating a fine-tuned CIM model (a rich business model) is vital to have a relatively smooth transformation into PIM and finally to make a fine-tuned PSM (Kriouile, 2015).

A CIM for an application (including geospatial applications) is generated through constant consultation with domain experts, business owners, and clients. It also requires direct communication with external partners and clients. According to the stakeholders' needs and requirements, different models could be used to describe the same reality in the application. In this section, the main objective is to find ways for designing business process models as the first stage in geospatial web application development.

As shown in the figure 3-5, CIM could be represented by two types of models: business process models or requirement models. According to Sharifi and Mohsenzadeh, 2012, different representations of the business process model could be classified into three types: UML Diagram, Data Flow Diagram (DFD), and Business Process Modeling Notation (BPMN). Also, The Requirement Model aspect can be classified into two types: Use Case Model and Feature Model.

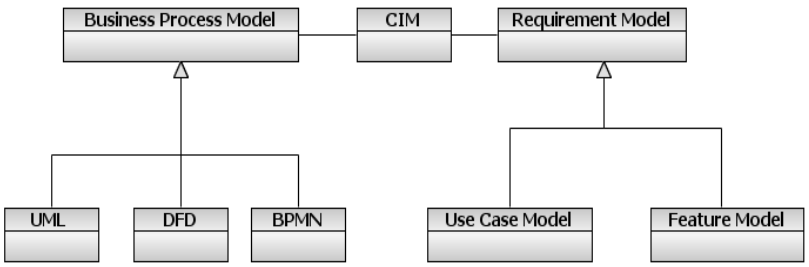


Figure 3-5- Taxonomy of CIM (Sharifi & Mohsenzadeh, 2012)

This research will effectively use the UML 2.0 activity diagram to present the business process model in order to achieve a concentrated CIM that (Rhazali et al., 2018). The activity diagram is another important behavioral diagram in the UML diagram to describe dynamic aspects of the system (Koch et al., 2002). The activity diagram

is essentially an advanced flow chart that models the flow from one activity to another activity. The following points should be considered when create an activity diagram:

- Identify candidate use cases through the examination of business workflows
- Identify pre-and post-conditions (the context) for use cases
- Model workflows between/within use cases
- Model complex workflows in operations on objects
- Model in detail complex activities in a high-level activity Diagram

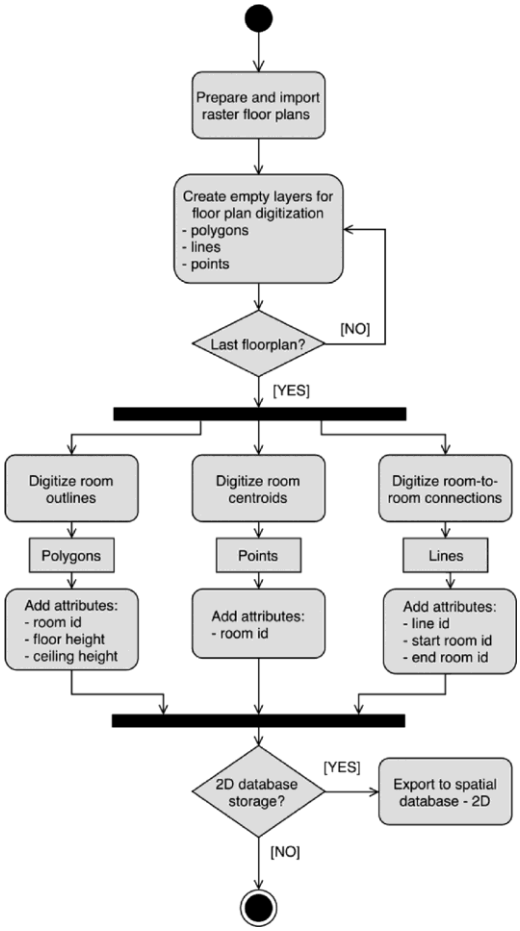


Figure 3-6- UML activity diagram for the digitization phase of a geospatial application (Tekavec & Lisec, 2020)

3.2.5. PIM

PIM is suitable to describe a domain problem (represented by CIM) in a proper information system. The most crucial point in PIM is its independence from a particular platform. There shouldn't be any details and technical specifications that could make PIM dependent on a specific platform. However, software engineers and developers have a more dominant role in creating PIM than CIM (Kardos & Drozdova, 2010).

As described, there are some limitations in using predefined modeling languages (such as UML) for PIM development and they cannot provide the required means to define PIM in several domains. Also they challenging to understand for domain experts and stakeholders with limited backgrounds in computer modeling (Kriouile, 2015).

The meta-object facility (MOF) specifications developed by OMG (OMG, 2011) allow developers and domain experts to define their modeling language by using the metamodeling concept (Ahmed, 2013). This

way, in MDA, before creating PIM and PSM, a metamodel should be created for each one of them. Metamodeling is the first step to create domain-specific languages (DSLs) for PIM. Since DSL is all about a specific domain, the domain knowledge should be captured and well maintained in defining DSL (Gorton, 2011; Osis & Nazaruka, 2008).

1- The modelling process depends extremely on the level of knowledge and experience of modeller
2- Based on the modeler's personal view, it might be possible to develop more than one PIM for the same problem
3- The main criteria are that how much model design help developers to achieve their goal
4- Depends on domain experts and their collaboration with developers
5- There is no bad PIM, however some models could represent the information system better than others
6- There are no widely accepted standards for creating PIM metamodel, the way to do the modelling mainly depends on the modeler
7- This is a continues process, more projects and more editions, more perfect PIM metamodel

3.2.6. PSM

PSM is a model of the information system that is more specific about using certain platforms or applying a particular technology for the development process. PSM has a lower level of abstraction than PIM (OMG, 2014). PSM will be created based on the MDA's model transformation concept. Unlike an ordinary MDA software development project, transformation in this research combines the ordinary PIM to PSM model transformation with specific transformation rules based on selected software design patterns. So, the resulting PSM is created for a particular platform and has the characteristics of those design patterns. The model transformation and creation of PSM from PIM is an automated process that uses transformation rules. These specific transformation rules are based on selected design patterns and created PIM to create and make a PSM.
In the MDA, PSM development should be done by experienced software developers and platform specialists and there is no need for domain experts to take part in this process.

3.2.7. Model transformations

Transformation is a fundamental theme in software engineering, and model transformation is one of the central elements of MDA. It refers to the automated way of modifying and creating new models from existing ones (Thomas, 2004). The MOF provide necessary concepts for developers and software engineers to define metamodels and transformation rules in the MDA process. Generally speaking, both input and output models of a transformation should have a distinct metamodel that confirms the MOF. Transformation rules that define model transformation come from a metamodel that describes transformation language, and transformation language itself should confirm the MOF (Kriouile, 2015; Marcos et al., 2003).

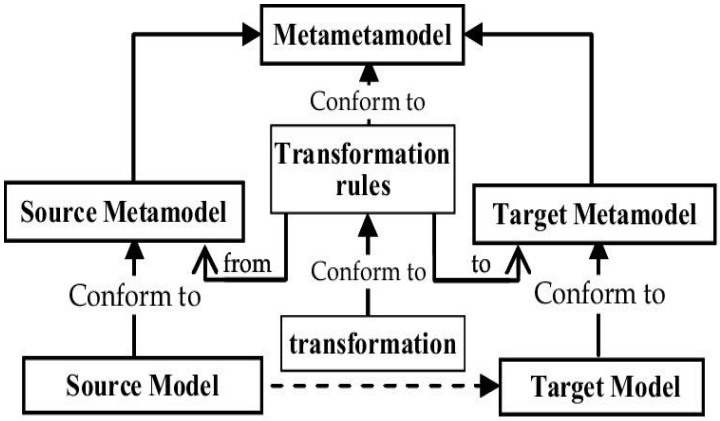


Figure 3-7- Integrating design patterns into MDA (OMG, 2003)

The most important applications of model transformations in the MDA process are (Gorton, 2011; Liu & Wang, 2011; OMG, 2003, 2014; Soley & OMG Staff Strategy Group, 2000):

- Producing lower level models from higher-level models and finally generating code from lower-level models
- Once the transformation is done (developing mappings and transformation rules) and there is no change in the information system, it could be used several times for different models and applications.
- To generate query-based views of the whole software system
- Managing tasks related to model evolution such as upgrading models or model refactoring in case of introducing changes to the system
- Facilitating reverse engineering of high-level models from low-level models and code
- It reduces the amount of time and energy needed to software code, basically by introducing an automatic and providing the possibility of model reuse.
- helps development process to support multiple technologies, platforms, and programming languages
- More engaging other stakeholders such as domain experts
- The possibility to use software design patterns and create a more consistent product
- Model-based software design allows the information system to be more flexible toward changes and updates, easily integrate them (OMG, 2014)

There are mainly two types of model transformations in MDA. CIM to PIM transformation and PIM to PSM transformation. CIM to PIM transformations are usually done manually (the process is not automatic). However, there are several attempts to (fully or at least partially) automate this process. This means still there is no fully functional machine-readable approach to map CIM elements into PIM (Sharifi & Mohsenzadeh, 2012).

PIM-to-PSM transformation is the most complex part of the whole MDA process. Technical experts mostly do this transformation with no or, in some cases, minor contributions from the project's clients. Model transformations in the MDA also could be used as bridges to transform to different PSM models (Kardos & Drozdova, 2010; Meservy & Fenstermacher, 2005) or, in some cases, to help to create a model with high level of abstraction from models with the lower abstraction (reverse transformation).

3.2.7.1. CIM to PIM model transformation

The main goal in CIM to PIM transformation is to convert the business model expressed by the UML activity diagram into PIM expressed in the class diagram or represented as a DSL. This transformation should be done using the developed CIM (the business model) and adding some technical information related to the information system and software design applying a set of transformation rules.

There have been several efforts to find a methodology for doing CIM to PIM transformation in a fully (or partially) automatic manner. However, manual transformation still is one of the most trusted and most accessible ways for this transformation.

Looking at the literature on CIM to PIM transformation, it could be seen that several efforts have been made to establish a structured methodology for this transformation. Considering the research scope and the fact that CIM to PIM transformation is one step in the whole MDA process and the limited amount of time, we decided to use the manual method for CIM to PIM transformation. The application's business logic (CIM) logic will be expressed using UML 2.0 activity diagram, and manual transformation based on a set of transformation rules proposed by Rhazali et al. (Rhazali et al., 2018) will be used to generate PIM.

One point worth noting is that the goal of this manual transformation is to Generate a metamodel (not a model) for PIM. It is slightly different from the actual methodology developed by Rhazali et al. For this thesis, the CIM to PIM transformation is about creating a PIM metamodel from the CIM model. to do this, the main rules of CIM to PIM transformation plus other considerations related to metamodeling creating DSL have to be taken into account.

The rules of thumb for CIM (detailed activity diagram) to PIM (metamodel) transformation are

- An object node translates into a class.
- The state of each object node becomes a class method (Rhazali et al., 2018)

We have to consider these two rules when dealing with and to try to define DSL for PIM. During creating a good CIM model and during CIM to PIM transformation, it is of high value to include domain experts, professionals, customers, and other stakeholders in the process, not just software engineers and developers (Rhazali et al., 2018). As the level of abstraction in the development process is reduced (from CIM to PIM, from PIM to PSM, and finally PSM to software code), software engineers and developers will be more involved, and domain experts will have more minor rules.

3.2.7.2. PIM to PSM Model transformation

Since introducing MDA as another software development paradigm, there have been several attempts to use it in the software industry and academic research (Meservy & Fenstermacher, 2005; Rahmouni & Mbarki, 2011; Rhazali et al., 2018; Schmidt, 2006; Thomas, 2004). Many studies and publications in this area focused on PIM to PSM model transformation to find an excellent approach to transform PIM to PSM in a fully (or partially) automatic manner. They have used different transformation rules, model transformation languages and also tried to apply them for different applications from health services to web development (Marcos et al., 2003; Rahmouni & Mbarki, 2011; Rodriguez et al., 2010).

To develop a web application based on the MDA principles, Huang and Chu tried to transform PIM into PSM automatically. They have introduced three types of mapping rules (namely boundary, control, and entity) for this transformation and a different set of transformation rules for generating source code from PSM in their application (Huang & Chu, 2014). However, their work on model transformation and the transformation rules definition is not precise, and it is not explained how they acquired those transformation rules and what kind of transformation methodology and languages they have used in their work.

In another work, Rahmouni and Mbarki proposed an approach for developing web applications based on the MDA process. Their goal was to increase the level of automation and increase development speed and decrease development-related errors. They have developed two metamodels for PIM and PSM in UML class diagrams and use UML notation for specifying transformation rules. Finally, they have transformed PIM into PSM using the ATL model transformation language (Rahmouni & Mbarki, 2011).

In their paper about model-driven development of information systems based on MVC (model view controller) architecture, Kazato et al. proposed a model transformation methodology based on MVC architecture. They have

used UML profiles to capture specific information system requirements and applied a set of transformation mappings to generate various PSMs (Kazato et al., 2009).
In MDA, PIM to PSM model transformation typically includes a set of transformation rules and two metamodels, one metamodel as input metamodel and the other as output metamodel. Based on the input metamodel, a model instance is also needed for the process. The final output of the model transformation is the model instance of the output metamodel generated by applying transformation rules to the input model (Ahmed, 2013). In this research, after developing a metamodel for PIM based on CIM transformation, PIM will be created. The next part is to define a metamodel for PSM based on the specific platform we want to develop our application and related programming language. Having two metamodels (one for PIM and one for PSM) and an instance of PIM metamodel, QVT transformation language will be used for PIM to PSM model transformation and automatic generation of PSM.

3.2.7.3. Model transformation languages

In MDA, software development uses the metamodel concept to reflect changes in information systems and models. The MDA model transformation provides necessary means for automatic (fully or partially) transforming the input model (with higher level of abstraction) into the output model (with lower level of abstraction). This way, the output model instance will be updated in case of any change to the input model. This is the best way to ensure some level of consistency in the system models (J. Saraiva, 2013).
The most crucial role of model transformation is to reflect the relationship between input and output metamodels using transformation rules. This way, any changes or updates in the information system and thus in input metamodel will be reflected in the output metamodel (J. Saraiva, 2013).
There are three main model transformation approaches in MDA: The first one is OMG's QVT transformation language. The second one is the ATL transformation language. And the third one is MOF based model to text transformation.
The ATL (ATL transformation language) developed by AtlanMed research team is a model transformation language that provides a rather natural approach to model transformation. This transformation language is successfully integrated into Eclipse Modeling Framework's M2M (model to model) modeling project as a plugin (Rahmouni & Mbarki, 2011).
MOF model to text (MOFM2T) transformation is a transformation language developed by OMG, and its primary purpose is to generate code from models. It is mostly used to generate source code from PSM models in MDA, unlike QVT that is normally used for PIM to OSM model transformation (J. Saraiva, 2013).

3.2.7.4. QVT transformation language

One of the famous transformation languages is QVT (Query/View/Transformation), a standard set of languages for model transformation in MDA, defined and developed by the Object Management Group (Czarnecki et al., 2006).

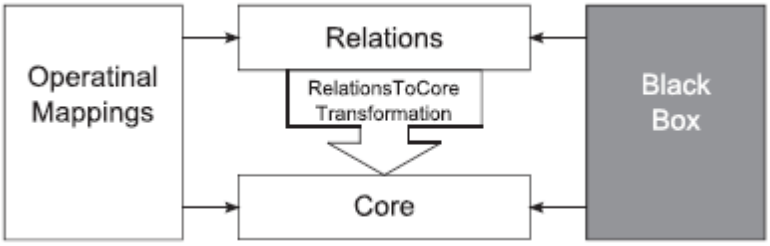


Figure 3-8 QVT model transformation (D. K. Kim et al., 2017)

There are three different domain-specific languages (DSLs) within the QVT: core, operational mappings and relations, and black box operations integrated with each of these DSLs (D. K. Kim et al., 2017).
Relations is a declarative transformation language that requires two models and a couple of conditions to define relations between model components. The core is also a declarative transformation language smaller than the others and does not use patterns. Often called as QVTo, operational mappings are imperative transformation languages that help to extend relations and core. Unlike relations which are bi-directional, operational mappings are unidirectional. Finally, the black box is an important part of QVT since it allows implementing complex algorithms in the desired programming language (D. K. Kim et al., 2017; Nolte, 2010; The Object Management Group, 2006; The Object Management Group., 2007).

3.2.8. Software development using the MDA

The architectural separation of concerns is the main foundation of MDA. Adaptation and successful implementation of MDA principles in web development help software engineers and web developers reach three crucial goals: reusability, interoperability, and portability (Schwinger & Koch, 2006).

There are several examples of the application of MDA for developing software and web applications in the last two decades. The information system developed by Caceres et al., 2003 is among the first systems developed based on MDA concepts (Huang & Chu, 2014). In the web application domain, the study by Tai et al. 2004 is among the first studies that shows the result of applying MDA to develop several a large-scale web application. They broke the main information system into smaller subsystems, developed a metamodel for it and use a team of developers to work on each subsystem individually. The result of study shows that using MDA and metamodeling specifications could increase the level of collaboration and communication in the project (Schwinger & Koch, 2006).
One example of using the MDA pattern in the geospatial domain and the public sector is using the model-driven development process by Danish Geodata Agency (DGA) for the integrated dissemination of national geographic data. The process starts from the conceptual modeling of the geographic data to the end, where data are distributed via WFS services and download capabilities (Huang & Chu, 2014). In a study by Aydinoglu and Kara, they examined the combination of MDA application development and linked data approaches in modeling and its impact on the dissemination of geographic data sets. This study used Turkey's administrative unit's geographical data (Aydinoğlu & Kara, 2019). The study results show that using linked data along with modelling and publishing geospatial data using MDA methodology, provides more opportunities for semantic interoperability, software reuse, and data accessibility.
Prakash and Kerry applied well-defined software design patterns to develop the display component of GIS software (Prakash & Karri, 2018). Their work focuses on finding suitable design patterns for recurring problems in developing GIS software. They explained the display component of a GIS application into several subsystems with various design problems and tried to use suitable design patterns to deal with each design problem (J. Saraiva, 2013).

3.2.9. MDA critics

While there are promising results in using MDA in software engineering and web development, there are a few issues and challenges regarding its functionality. One of the most important criticisms is about the prominent use of UML for modeling purposes. UML itself faces some challenges as a modeling language it is hard to understand and work with for several people (domain experts) without any background in computer science. The other issue with UML is its limitation to model beyond IT related domains such as biology or geoscience. However, this can be addressed using design-specific languages (DSL) in the development process.
The next one regards model and code maintenance and update. While the software code is generated automatically, it is not enough to build an application from it. In many cases the generated code covers less than 50% of the code

required application so lots of manual enhancements should be done to an automatically generated code (J. Saraiva, 2013; Thomas, 2004).

3.3. Pattern Based Model Driven Architecture (PBMDA)

As described in the introduction, the main objective of this chapter is to describe PBMDA methodology developed for geospatial web application development. The PBMDA developed in this chapter is based on principles of MDA and tries to integrate software design patterns into the MDA process. The most important part of this methodology is to find a way for this integration. While there are several types of research on design patterns and how to use them in an actual real-life problem (Buschmann et al., 1996; Dong et al., 2010; D. K. Kim et al., 2017; McArthur, 2008; Pang et al., 2011), there are few studies that look at design patterns and their implementation as part of the MDA software development process (Seffah, 2015; Taleb et al., 2007; Zadahmad Jafarlou et al., 2010). In the initial MDA specification published by OMG (OMG, 2003), OMG proposed using design pattern concept within MDA (figure 3-9). It suggested that the design patterns could be incorporated into the MDA process as part of PIM to PSM model transformation. That means a certain model transformation methodology that considers design patterns shoulb be applied to generate PSM from PSM.

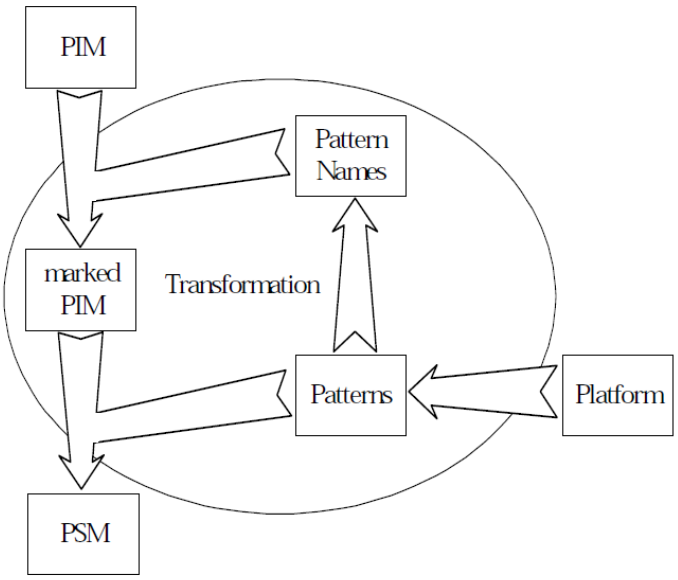


Figure 3-9- Incorporating Design patterns into MDA (OMG, 2003)

Seffieh (Seffah, 2015) proposed an approach for pattern-oriented model-driven architecture (POMA) and a way to integrate design patterns into the MDA development process. This approach enables software engineers and developers to deal with more complex and interactive systems. He suggested that these two programming paradigms could be incorporated at different abstraction levels and create a heterogeneous architecture for developing complex software.

The proposed system is combined MDA's regular models (CIM, PIM, and PSM) in 5 different architectural levels and the respective model transformations. As figure 3-6 shows the integration of design patterns to this structure happens as a set of mapping rules in PIM to PSM transformation. The PIM to PSM transformation includes two types of transformation: one is the regular transformation rules for PIM to PSM transformation. The other is specific mapping rules for transforming PIM (without any trace of design patterns) into a pattern-oriented PSM (Seffah, 2015). This proposed framework is also in line with the OMG's original for pattern integration into MDA process (OMG, 2003).

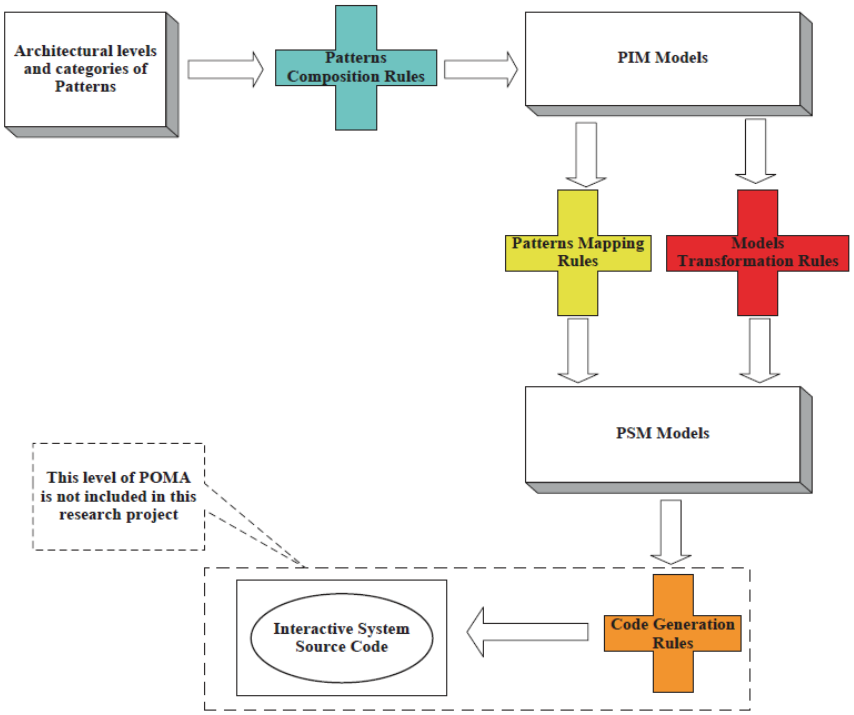


Figure 3-10- POMA architecture (Seffah, 2015)

Figure 3-11 shows the PBMDA methodology for developing geospatial web applications. PBMDA proposed in this research combines MDA principles and software design patterns based on POMA methodology proposed by Seffah, 2015 that is the same process as MDA with slight changes and enhancements in PIM to PSM model transformation.

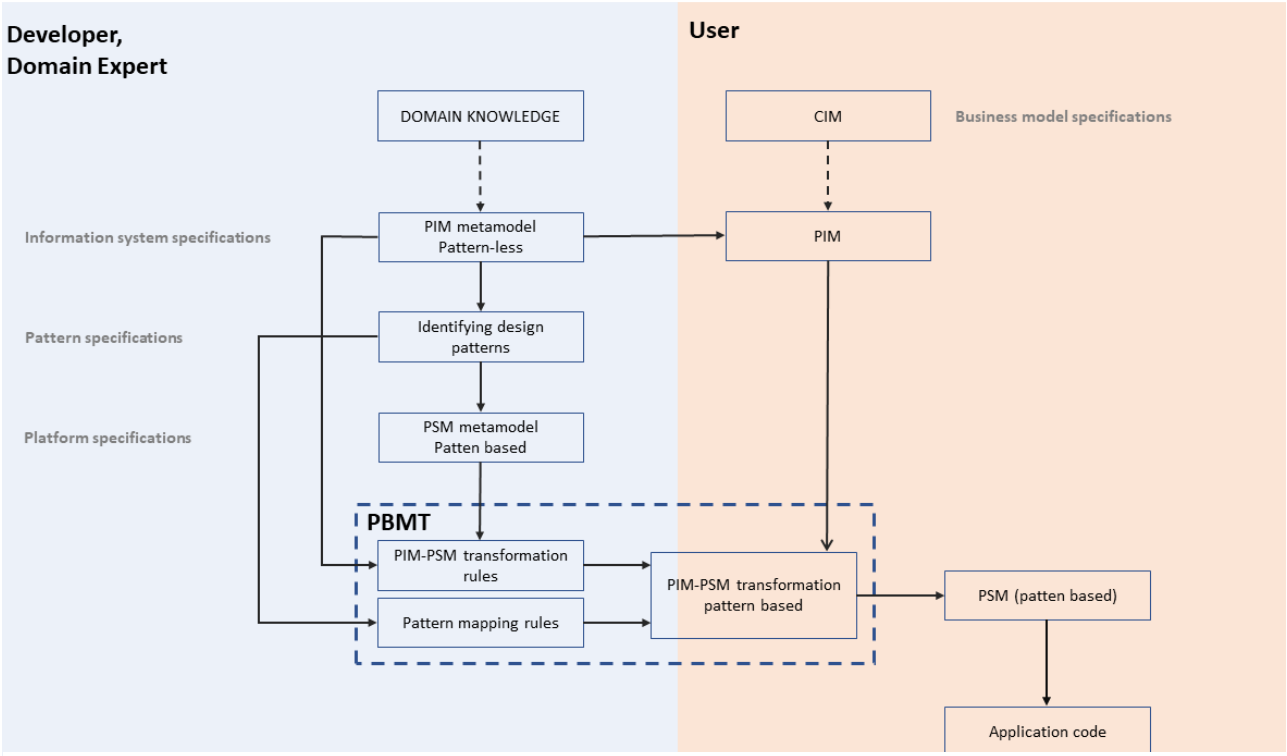


Figure 3-11- The PBMDA methodology

At first, it is very beneficial to divide the information system into several smaller and simpler subsystems, each subsystem with one specific goal. Then we can consider a separate viewpoint for each subsystem and use the PBMDA approach for its development. Viewpoints can be defined based on context and abstraction level. Each viewpoint can have its metamodel (DSL), models, and model transformations in MDA (Ahmed, 2013). Then it is possible to link those subsystems and create an integrated system. This way, the level of system complexity will decrease significantly since we only have to deal with one subsystem in the modeling process.
The first is creating CIM based on the application's logic and specific problems and related solutions to address them. There has to be constant consultation with all stakeholders in creating CIM, including organizations, domain experts, and end-users. The goal in this step is to reach some specifications to express the application's business logic. The UML activity diagram will be used to show the results.
The next step is creating PIM with respect to the resulted CIM. In PIM development, system design principles have to be considered, and domain experts have to be consulted regularly. Since UML is not suitable to capture all aspects and specifications of problems, processes, and solutions in our field (Geoscience), we need to define a DSL for this domain (since a functional DSL for this domain does not exist yet). It should be mentioned that the DSL created in this research is not a complete one, and it should be adjusted and enhanced many times to be a complete DSL and be used for other domain projects. The PIM metamodel (DSL) has to be generated mainly by domain experts with a small contribution of developers. PIM is a model instance of PIM metamodel and has to be generated after creating PIM metamodel based on the project specifications.
The next step is to generate a metamodel for PSM, considering all the specifications related to a particular platform and the technologies we want to use in the implementation phase. PSM metamodel has to be developed mainly by software developers and platform specialists based on project needs and requirements. It is also possible to develop more than one PSM from a PIM each PSM designed based on a particular platform specification.
Creating PIM and PSM metamodels is essential to investigate them and identify the possibility of using specific design patterns in certain parts of them to generate a pattern-based PSM metamodel from a pattern-less PIM metamodel. This way, it is possible to use Pattern-Based Model Transformation (PBMT) for PIM to PSM transformation. The PBMT will be explained in detail in the next chapter. This transformation will be done using QVT transformation language. Two sets of transformation rules have to be defined for this transformation, ordinary transformation rules for transforming PIM into PSM and pattern mapping rules for design patterns.
Defining PBMT for PIM to PSM model transformation, by introducing PIM and PSM metamodels and PIM model instance to the transformation function as input, we can have PSM model instance as the final output of the transformation. The final part is to generate application code from PSM.
This approach will be implemented in this thesis as the way to integrate design patterns into MDA for developing a geospatial web application. The following section is dedicated to PIM to PSM model transformation and pattern-based model transformation (PBMT).

3.3.1. Pattern-Based Model Transformation (PBMT)

Design patterns are one of those concepts in computer science that are somehow challenging to use in the software development process. Many software engineers and developers mainly rely on their experience for a successful implementation of design patterns (D. K. Kim et al., 2017). however, it is possible to use design patterns in various parts of the development process in the MDA, from PIM to PSM model transformation and even in code generation phase. In recent years, design patterns have been used by many academics and field professionals for developing complex software systems (Pang et al., 2011). Design patterns are great tools that increase software quality by bringing an acceptable solution for a common problem in software design (D. K. Kim et al., 2017).
By incorporating design patterns into the MDA software development process, software engineers and developers are able to use software development units within MDA. This will increase the modeling granularity in MDA and allow to build of extensible and more flexible information systems (Pang et al., 2011).

Several studies have shown how to implement design patterns into model transformation, and many researchers proposed different approaches for pattern-based model transformation (D. K. Kim et al., 2017; Pang et al., 2011; Seffah, 2015). However, it is worth mentioning that this transformation is not a simple task. Many factors should be considered, like pattern detection, automation, transformation rules, and transformation language.

Briand et al. worked on developing automatic pattern detection techniques in software models and applying design patterns into software models. Their work emphasized pattern applicability and how it can be found that a design pattern is suitable for a particular model (Briand et al., 2006).

El Boussaidi and Mili tried to identify design problems in a development model in their work on software design patterns. Then they tried to apply appropriate design patterns into the software model to solve those problems related to system design. Finally, they have transformed the information system's model based on the identified patterns and using a set of transformation rules (el Boussaidi & Mili, 2012). They used two types of information models: the problem model (input) and the solution model (output). They transformed the problem model into the solution model using a set of transformation rules based on the applied design patterns in the Eclipse Modeling Framework (EMF).

In another initiative, Dong et al. used QVT model transformation language for applying design patterns into the software development process. They transformed a problem model (a model with no trace of design patterns) based on a set of design patterns to a solution model. They imported pattern solutions as input model, defined transformation mappings, and used them in QVT language to create a new model. However, they only focused on class diagrams in their work (Dong et al., 2010).

To implement the design pattern in our thesis, we will use pattern-based model transformation approach introduced by Kim et al. (D. K. Kim et al., 2017). In pattern-based model transformation (PBMT), a problem model will be transformed into a solution model using a set of transformation rules (D. K. Kim et al., 2017).

In PBMT, each design pattern will be described as a set of specifications related to the problem, solution, and transformation. This will happen at the metamodel level (M2). The problem specifications are actually a set of problem models that could be addressed using solution specifications or design patterns (D.-K. Kim & el Khawand, 2007). Each solution specification is designed based on a particular design pattern and corresponds to a problem specification. Transformation specifications in this approach include rule bindings between problem and solution specifications and related transformation rules (figure 3-12). They explain how a particular problem specification should be addressed by its equivalent solution specification (D.-K. Kim & el Khawand, 2007).

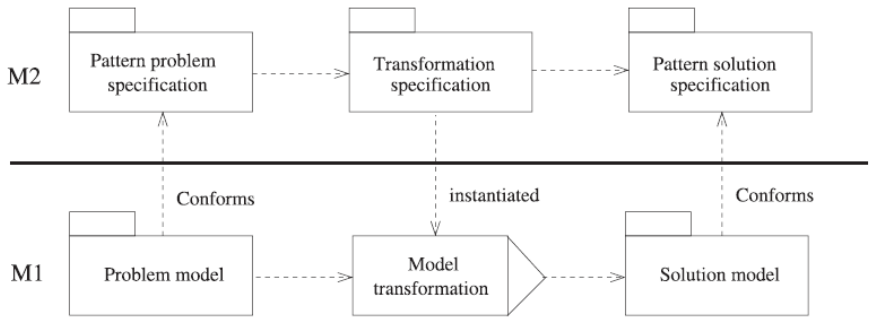


Figure 3-12- Pattern based model transformation at metamodel level (D. K. Kim et al., 2017)

QVT transformation is the model transformation language that helps us to define transformation rules and transform the input model (with the problem specifications) to the output model (with the solution specifications). This whole approach is developed based on the metamodeling concept where every design problem, design solution (which is somehow design patterns) are defined at the metamodel level (M2), and the application model has to be defined at the model level (M1).

One of the main objectives of this research is to present a systematic methodology to develop an application by using design patterns in the MDA model transformation to increase software quality during the design process.

This research will try to demonstrate this approach by applying PBMT for the observer design pattern (Gamma et al., 1995) in our final application.
The reason for choosing the observer pattern as the candidate pattern for PBMT is that this pattern is widely used in many software development projects, so showing its application in the PBMT process could be helpful for other researchers and developers. Also, the main objective of this research is to represent an approach suitable for integrating design patterns into the MDA process. For showing the effectiveness of research methodology, one design pattern (in this case, the observer pattern) was needed to be used in PBMT as part of the research methodology. The procedure to identify the use of design patterns in the information model and choose suitable patterns to be used in the PBMT and other issues like pattern applicability and pattern conformance is out of the scope of this research and should be addressed in other studies.
Observer design pattern (Gamma et al., 1995) is a well-known example of software design patterns. As shown in figure 3-13, the Observer design pattern solves the issues related to the management of state dependencies in software systems by introducing callback interfaces and registration (Riehle, 2011). During the application of this design pattern, software engineers and developers use specific terms like "observer" and "subject" as part of their language to communicate the design, implementation, and documentation of the software development process (Riehle, 2011; Riehle & Züllighoven, 1996; Taleb et al., 2007).

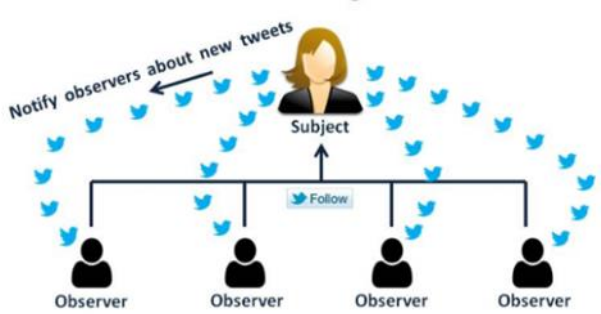


Figure 3-13- The Observer design pattern

First thing to do for PBMT in this thesis, is to analyze the PIM metamodel and identify where we could apply a certain design pattern (the observer pattern). There are several publications related to how the pattern applicability could be done. However, since in this thesis only one pattern once will be applied to the PIM metamodel, this will be done manually and the part of metamodel (those classes) suitable for observer pattern application will be identified.
After a brief explanation about the observer design pattern, it is time for presenting this pattern as a set of problem specifications and solution specifications. After introducing the solution specifications and the problem specifications, their respected metamodels should be expressed. These metamodels are going to be used in the transformation. the UML will be used to represent those metamodels because of its compatibility with the QVT transformation notation and its ease of use. Figure 3-14 shows the problem specifications of the observer design pattern and its respective metamodel in UML.

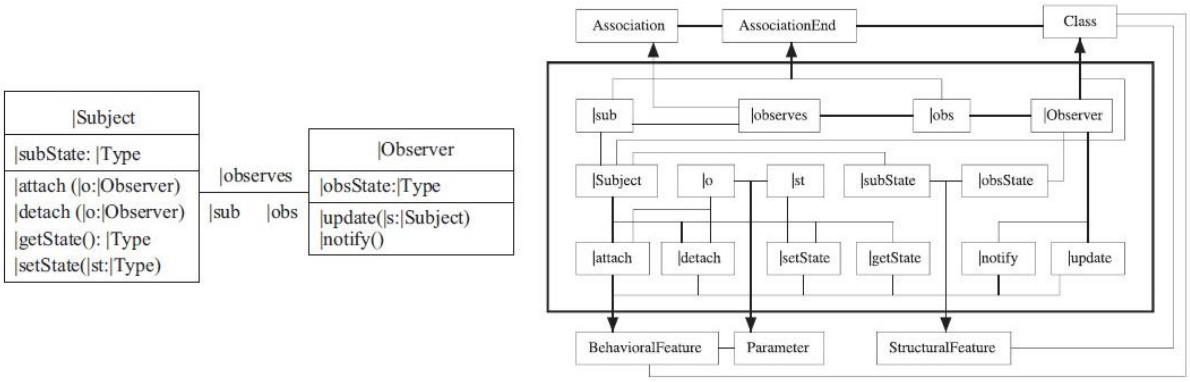


Figure 3-14- Observer pattern problem specifications (left) and corresponding metamodel (right) (D. K. Kim et al., 2017)

Figure 3-15 shows the observer design pattern solution specifications and its respective metamodel in UML.

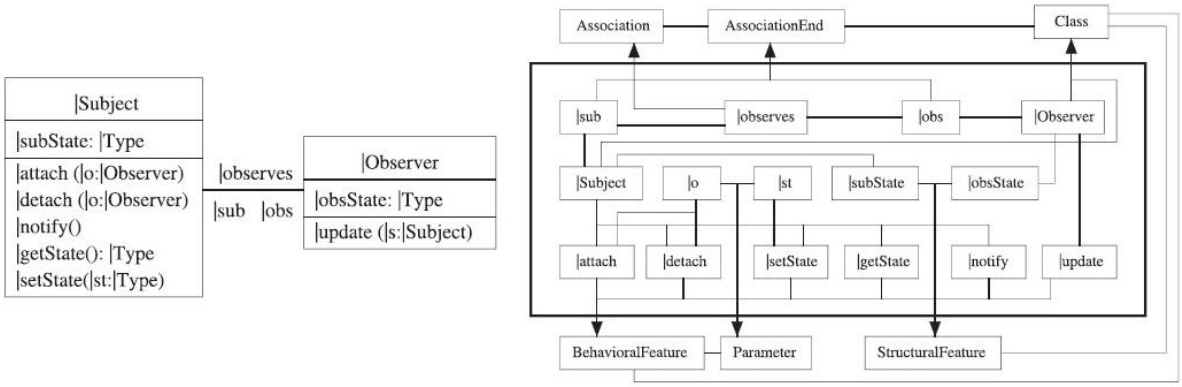


Figure 3-15- Observer pattern solution specifications (left) and corresponding metamodel (right) (D. K. Kim et al., 2017)

By looking at the observer pattern's problem specification and solution specification, it is clear that these specifications conform to their respected metamodel presented in UML. Also, it is recognizable that the solution specification is designed to help developers to address the design problem specified in the problem specification through the application of the Observer design pattern notation (D. K. Kim et al., 2017; D.-K. Kim & el Khawand, 2007).

After identifying design patterns, specifications, and their respective metamodel, it is time to incorporate these patterns (e.g., their problem specification and solution specification) into the MDA development process. The best way is to integrate the pattern problem specification into PIM and the pattern solution specification into PSM. It is vital to consider other aspects of MDA when doing this integration.

For example, when we want to integrate solution specification into the PSM, we have to consider the specific platform that we want to develop the application, so there should be specific changes to make solution specifications ready for integration with PSM, and it has to be designed based on the platform in use.

The next part is to define transformation rules based on the problem and the solution specifications. These pattern-based transformation rules have to be defined for transforming PIM (with problem specifications) to PSM (with solution specifications) and are a part of MDA transformation rules. Based on the specific pattern selected for this research (the observer pattern) and corresponding problem specifications and solution specifications, three general transformation rules have to be defined "problem-to-solution-subject," "problem-to-solution-observer," and" problem-to-solution-observes-association."

The following image illustrates the "problem-to-solution-subject" transformation rule and how it translates a problem specification into a solution specification. As it can be seen from the image, in the observer design pattern,

the subject (which is observed by the observers) gets a new method (notify()) which allows it to notify observers when there is a change in its state.

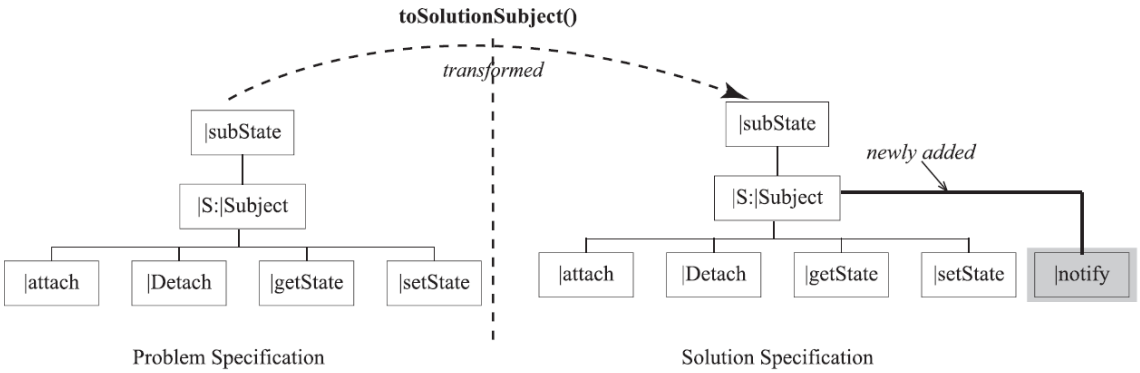


Figure 3-16- The Observer pattern, toSolutionSubject() operation (D. K. Kim et al., 2017)

The following image illustrates the "problem-to-solution-observer" transformation rule and how it translates a problem specification into a solution specification. As it can be seen from the image, in the observer design pattern, observers (which observe the subject) no longer need to note the subject for any updates constantly, and there is no need for notify() method because, in case of any update, the subject will notify them.

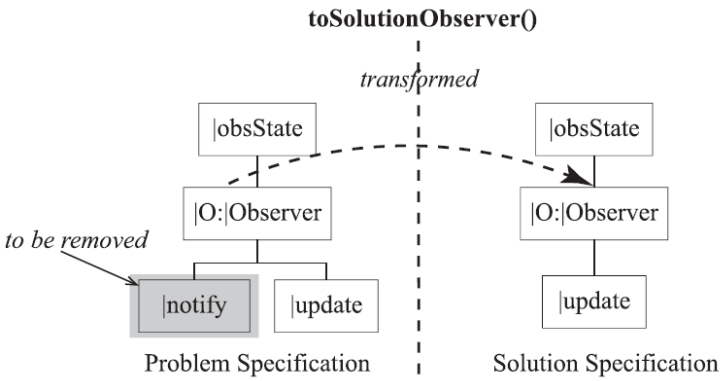


Figure 3-17- The Observer pattern, toSolutionObserver() operation (D. K. Kim et al., 2017)

After integrating the problem specifications and the solution specification into PIM and PSM metamodels respectively and defining pattern-based transformation rules based on the specified pattern, it is time to integrate this PBMT into the MDA model transformation and implement it using QVT transformation language.

3.3.2. Developing PIM metamodel

As it was discussed in the previous section, first we have to define a metamodel for PIM. The PIM metamodel should be defined with respect to the application's characteristics provided by domain experts and the principles of metamodeling and DSL generation.

Here are the main statements regarding to the CIM in order to create PIM:

- In the application there are three main parts: database, back-end, and front end (user interface)
- The application could have one or more databases. Each database has one or more data collections, and each data collection consists of one or more datasets.

- The application could have one back-end component. The backend component includes several backend services, each backend service works with one dataset.
- The application should have at least one front end component. Each front end component has a navigation bar and includes several sub-components (in this case map component, table component, and form component)
- The component gathers user's search information from it's subcomponents (e.g., form component) and sent it to the backend service. Backend services transform it to a request, send it the dataset, and receive the response from the dataset. Then send the response to the front-end component. Based on this response, front end component updates itself and its subcomponents.
- Each sub-component gets information from the main components and updates itself. Then displays relevant information by using web services (for example map component shows location information using WMS, WFS).

The PIM metamodel development is done using EMF. Since this research don't use a particular automatic methodology for CIM to PIM transformation, in Creating PIM metamodel, developers have to constantly check the resulted metamodel with the CIM and make sure they match. The developed PIM metamodel gives us a language and provides a framework for us to define PIM. It means that the PIM always confirms to its metamodel. Appendix A provides the necessary information and explains working steps regarding creating a modelling project and a metamodel for PIM in EMF. Also, you could find the generated metamodel in XML format in Appendix B.

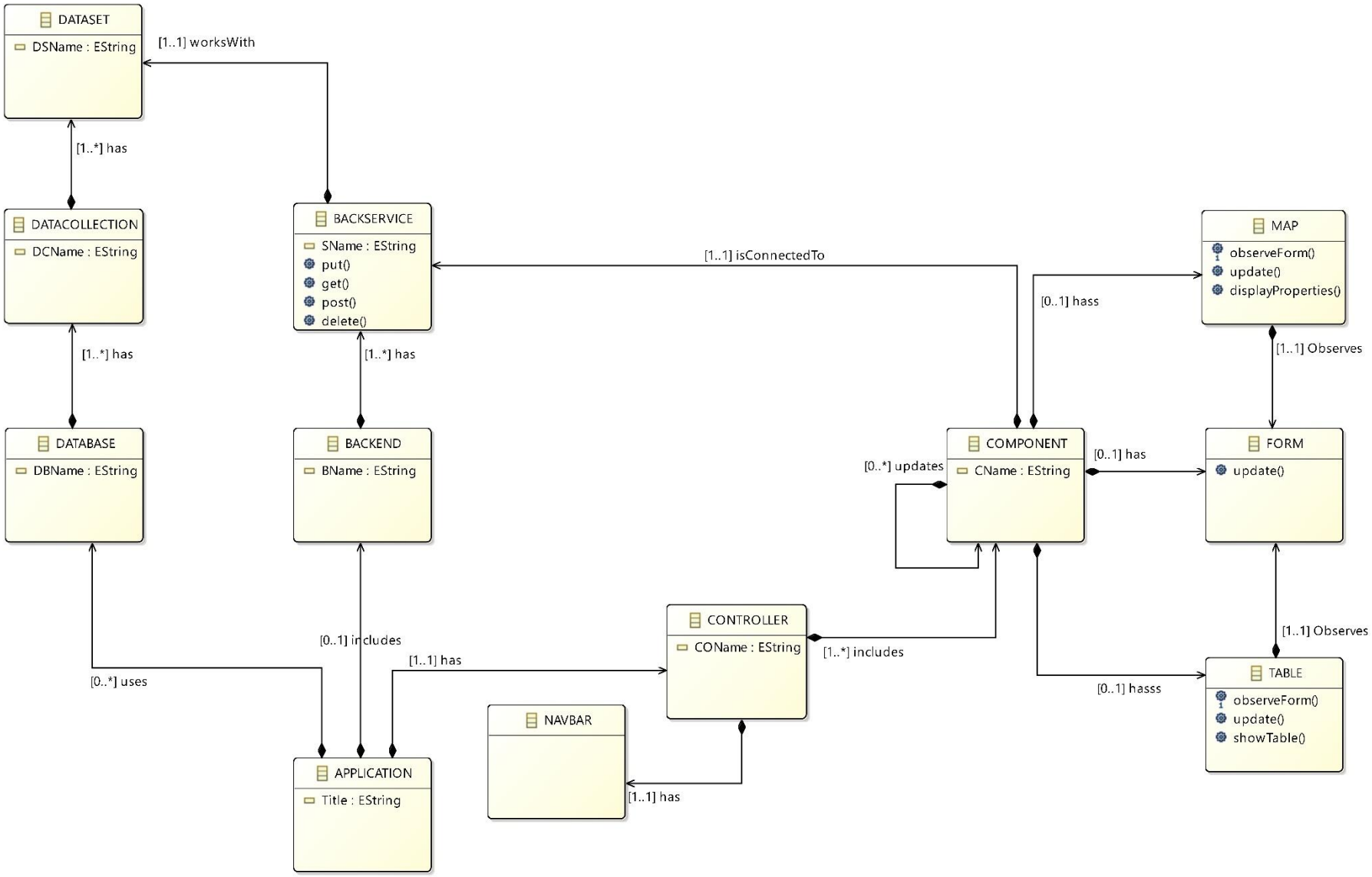


Figure 3-18- PIM metamodel

3.3.3. Developing PSM metamodel

Creating the PSM metamodel and the PIS to PSM model transformation are the most complicated tasks in the MDA software development process. Usually, experienced software engineers and platform specialists are in charge of developing PSM and doing PIM to PSM model transformation since these tasks require professionals with knowledge and experience of software architecture and software development on particular platforms.
The PSM metamodel has to be generated concerning the project's platform specifications and technical requirements, the generated PIM metamodel, and suitable software design patterns to use. The requirements and specifications of the application have to be introduced to the MDA process after developing PIM.
The choice of technology for the project's implementation and its platform specifications usually depends on the project goals and objectives. For example, some programming languages and their respective platforms are lighter and faster than others, more suitable for scalable projects with many interactions between users. Some are more suitable for heavy processing tasks dealing with various datasets. While these platforms can provide high performance, they cannot be used for scalable and fast projects.
Aside from the platform performance, scalability and efficiency, feasibility (in terms of time and labor needed to reach the desired functionalities), the knowledge and experience level of the software development teams play an essential role in choosing a platform for the project implementation. It should be reminded that platforms should help us reach project goals and provide tools to support desired functionalities and performance.
For this project, implementing the designed geospatial web application in MDA will be entirely based on the JavaScript programming language. Here you could find the related platforms that support almost all the desired functionalities in the project. Based on these specifications, PSM should include all the necessary functions, methods, and classes for project implementation.

- Programming language: JavaScript
- Front-end platform: React JS
- Backend Platform: Express JS (Node JS)
- Database: MongoDB

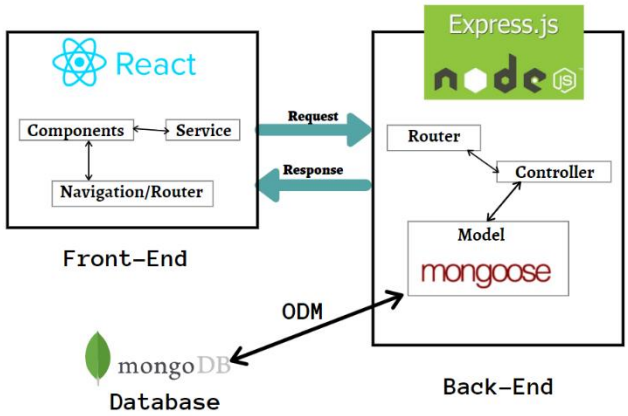


Figure 3-19- The application development stack (https://medium.com/techiepedia/what-exactly-a-mern-stack-is-60c304bffbe4)

In developing PSM metamodel, the last thing to consider is software design patterns and how to use them in the PSM design. To develop a pattern-based PSM metamodel, the first thing to do is identify the opportunity to use design patterns in the PIM metamodel (as a set of problem specifications). We have to look at the PIM metamodel and identify those parts of the PIM metamodel (those classes and relationships between them) that could be considered a pattern's problem specification. Then try to change those parts using the pattern solution specification. We have to apply design patterns in PSM metamodel so the PSM metamodel would be based on solution specifications.

As it discussed in the previous chapter, in this thesis we are going to use the Observer design pattern as an example to show how PBMDA works. The first thing to do is to identify problem specifications related to the Observer design pattern in the PIM metamodel. Figure 3-20 shows the problem specifications related to the observer design pattern in the PIM metamodel.

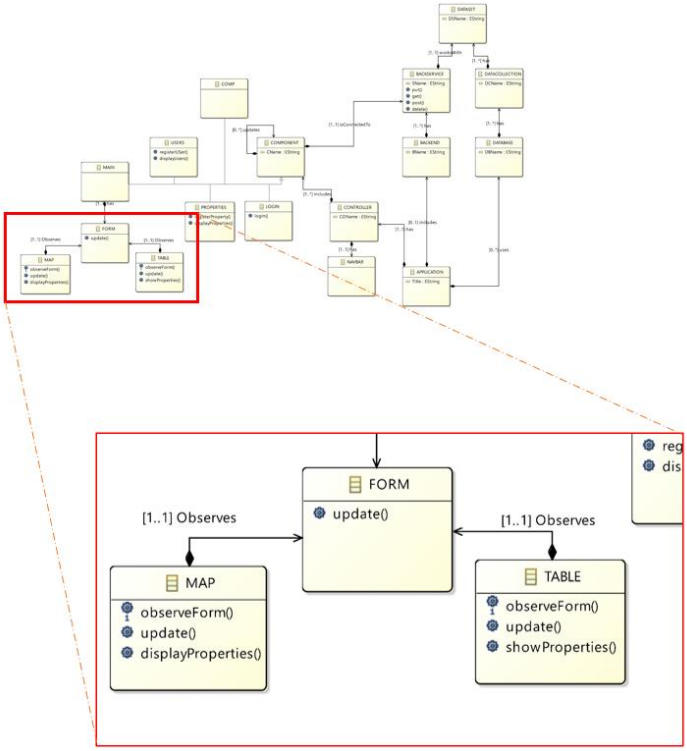


Figure 3-20- The Observer pattern problem specifications in PIM metamodel

Based on the provided platform specifications, the generated PIM metamodel, and the pattern problem specification in the PIM metamodel, we start to generate the PSM metamodel. The procedure to create the PSM metamodel in EMF is the same as creating a metamodel for PIM, but this time with new classes, attributes, and methods. Figure 3-21 shows the provided PSM metamodel. Appendix C shows this metamodel in the XML format.

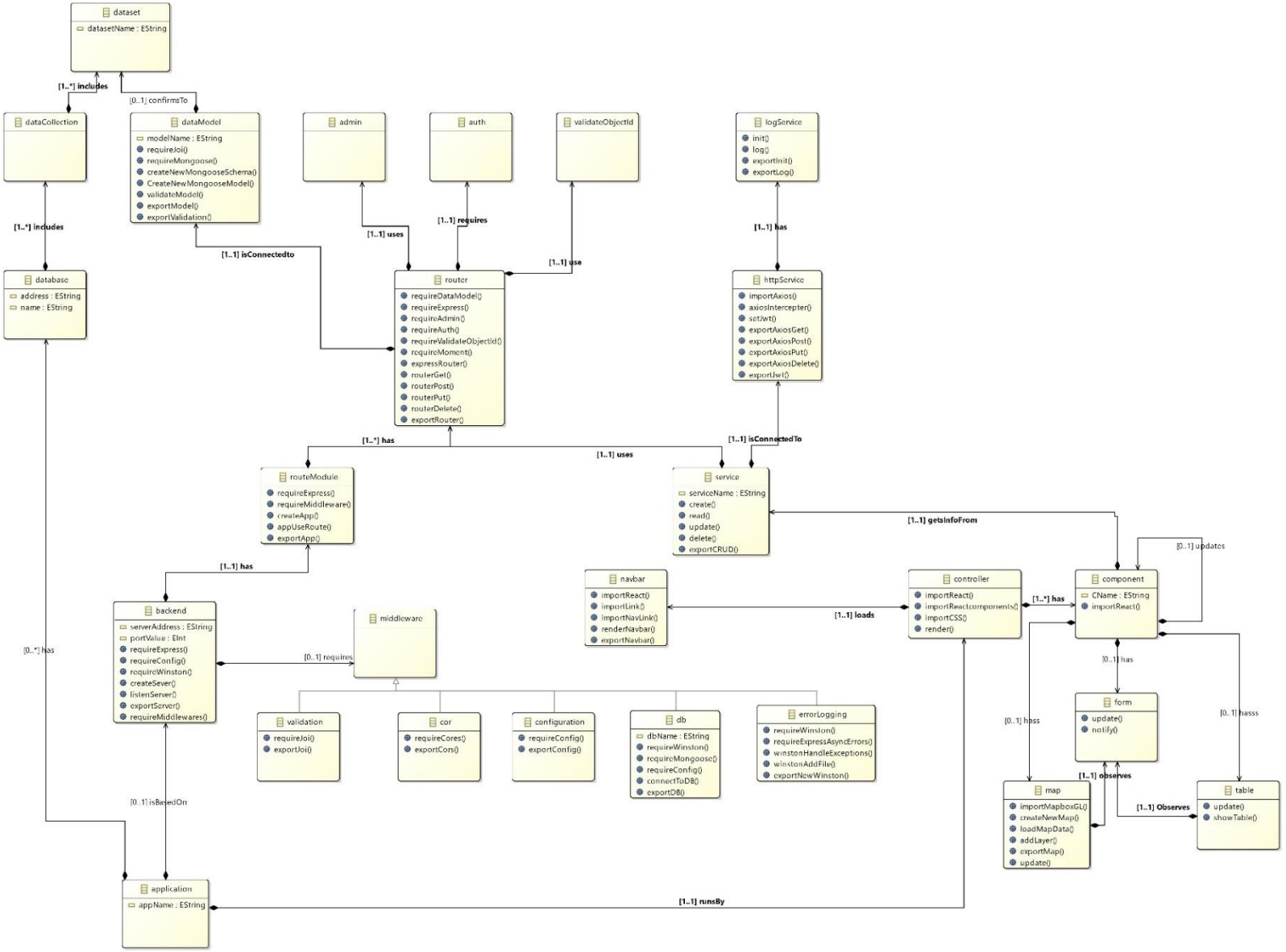


Figure 3-21- The developed PSM metamodel

4. RESULTS AND CONCLUSIONS

4.1. Introduction

This chapter could be a good starting point for those willing to work on MDA-based geospatial software development dealing with concepts like modeling, metamodeling, and model transformation. In this chapter, a geospatial web application (with one or more functionalities) has been developed for implementing the research methodology. The idea is to develop a standard geospatial web application and characterize it using the PBMDA methodology introduced in the previous chapter. So, it is essential to understand specific problems and issues in developing web applications and try to solve them by applying the PBMDA methodology.

At first, a short introduction about the tools and framework for applying the PBMDA approach has been provided. This research uses Eclipse Papyrus and Eclipse Modeling Framework (EMF) as the main toolsets for implementing the research methodology. So, some explanations on each tool and its essential characteristics are provided.

After explaining the research tools and frameworks, it is time to define and a geospatial web application as the case study to showcase the research methodology (PBMDA) introduced in chapter three. After providing some essential explanations about this geospatial web application, such as its goal, characteristics, and main features, the application has to be divided into different viewpoints, and one viewpoint (such as data management, data visualization, or data analysis) has to be selected for the application development.

The next step is to apply the PBMDA methodology to develop a web application for the selected viewpoint. At first, A CIM will be created for the selected functionality using UML 2.0 activity diagram notation. Creating the PIM from CIM would be the next step. The PIM will be generated using the PIM metamodel developed in chapter 3 and the CIM developed in this chapter. The next phase is PIM to PSM model transformation. A set of transformation rules have to be defined to convert PIM into PSM. These transformation rules have to be designed based on the suitable design patterns (for the PIM), mappings between PIM and PSM metamodels (developed in chapter 3) and considering the final web development platform (programming language, web framework). The PSM is the result of this transformation and an essential outcome in the whole PBMDA methodology.

After applying transformation rules and creating PSM from PIM, it is time to generate platform-specific code from the PSM model. The generated code from this step is the actual code and is used for the application development. Although the code generation process is supposed to be automatic, it always needs some manipulations and editions. The generated code is not enough to cover all aspects of desired functionality and usually requires manual enhancement. The last part is the manual enhancement of the generated code and making sure that it is suitable for the final application.

4.2. Modeling toolset

Since one of the main objectives of this research is to develop a geospatial web application based on the PBMDA methodology, metamodeling, modeling, and how the models are created, manipulated, and transformed is highly important to the research. Based on the developed methodology, a set of software modeling tools and frameworks have to be used in this thesis. The main criteria for choosing tools and frameworks are open-source, widely used, and available support in the software community. As a result, the Eclipse Modeling Framework (EMF) has been used as the primary toolset to implement this methodology for creating models and metamodels (PIM and PSM) and doing the transformations. Also, Eclipse Papyrus has been used for creating the CIM model based on the UML 2.0 activity diagram.

4.2.1. Eclipse Papyrus

Papyrus is an open-source modeling environment developed based on Eclipse. This tool could be used as a stand-alone tool or as a plug-in in EMF and allows developers and software engineers for graphical modeling according to several modeling standards, including UML 2. Papyrus is an extensible modeling tool with the possibility to support Domain-Specific Language (DSL) and Systems Modeling Language (SysML). The adapter allows to trace into such models and extract artifacts and links. For this research, Papyrus is used for developing CIM in the UML 2.0 activity diagram.

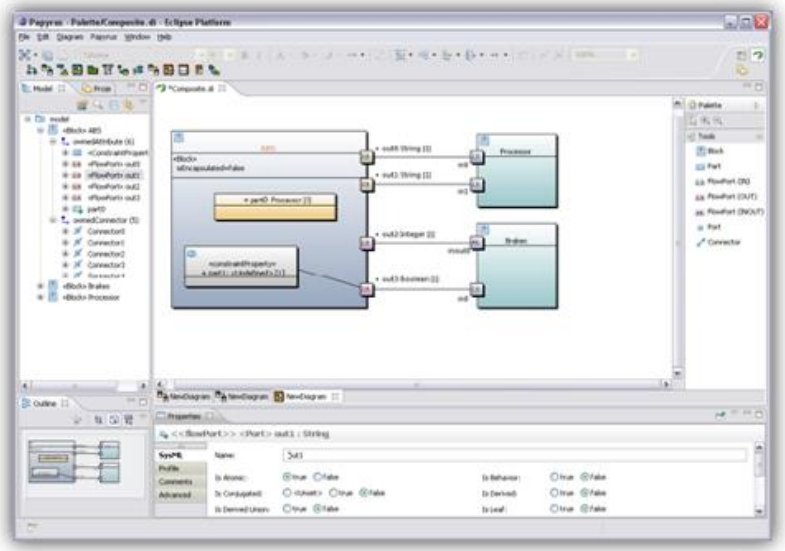


Figure 4-1- Papyrus User interface (eclipse.org)

4.2.2. Eclipse modelling Framework (EMF)

The Eclipse Modeling Framework (EMF) project is an open-source modeling framework for all sorts of development-related works, including modeling, metamodeling, model transformation, and automatic code generation. This framework has many built-in modeling facilities, and several plug-ins enable developers to create different kinds of applications for different purposes. This framework is highly suitable for developing tools and applications (software applications or web applications) based on a structured data model. So far, several applications and DSLs have been developed using EMF, and today more developers are using EMF in their projects. Its vast ecosystem allows developers to use it for developing a wide range of complex software and applications.

The Eclipse Modeling Framework (EMF) is used as the primary toolset for dealing with models, metamodels and transforming them for this research. The main reason for using EMF as the primary tool for developing a geospatial web application based on PBMDA is its built-in capabilities to work with OGC standards like (MOF, QVT model transformation) and its ability to work with metamodels and DSLs. Also, this framework is supported by a large and very active open-source community.

EMF is using XMI (XML metadata interchange) for storing and communicating models and metamodels. The modeling and EMF structures are based on three main concepts: core, edit, and codegen. EMFcore is the most important component in the EMF. It is developed based on the OMG's MOF notation. It also helps developers create and manipulate metamodels (DSLs or UML). Also, it is the main base for model transformation within EMF. EMFedit helps create and display model instances, model validation, and other operations and services and builds editors and model views. Finally, EMFcodegen is automatically

generating platform codes from models and providing code editors. The following picture shows the necessary steps to create and run a model in EMF.

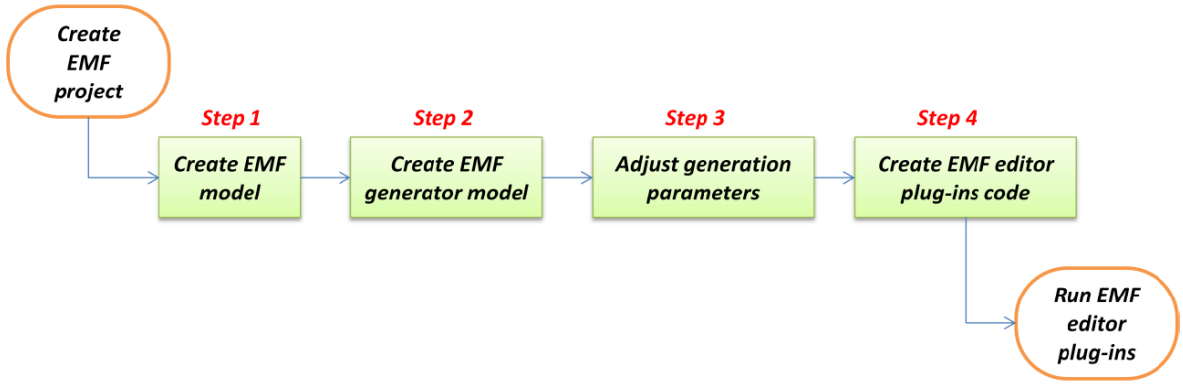


Figure 4-2- the process to build metamodel and generate model in EMF (Ahmed, 2013)

4.3. The application

Based on the descriptions provided in the previous sections, the case study application has been developed using the PBMDA methodology, and related models (CIM or the application's business logic, the PIM, and PSM) are defined or generated through this process. However, before starting the application development, it is better to introduce the geospatial application (the research case study), describe its main characteristics, and explain some of its functionalities. Also, it is crucial to choose one specific viewpoint of the application and focus development work on it.

In this section, the application and its specific development viewpoint are described. This process (to define and explain the main application and choosing one viewpoint) has to be done jointly by software developers and domain experts concerning the specific domain problems and the provided solutions to address them. This way, they can avoid several different issues simultaneously and reduce a great deal of software development complexity (by dividing the application into several relatively autonomous components).

When describing the application and its specific viewpoint, there is no need to add any technicality regarding application development, desired programming languages, and frameworks. The application logic has to be explained in simple plain text. The goal here is to express the application's primary purpose, components, and workflow. However, domain experts should provide these explanations, and they have to consider all the clients, end-users, and stakeholders. There are some formats (forms or structured questionnaires) to identify critical aspects and principles of the application. However, there is no use for such a questionnaire in this research, and everything required for its development is explained by text in this section.

In this research, a property rental application is selected as the geospatial web application to be developed using the PBMDA. The application's primary purpose is to help users find a particular property (house, apartment, studio, room, or land parcel) to buy or rent based on the user's specific criteria (money, property size, location, and other characteristics of the property). The user can search for a property based on different spatial and non-spatial queries. The specific viewpoint selected for development in this research is focusing on visualizing information (based on a query). By focusing on this viewpoint, there is no need to deal with other unrelated issues like database, storing data, or application's search logic.

Here are some descriptions about this particular viewpoint of the app (for showing the data):

- When the user opens the application, she/he will see a webpage including four different tabs: Main tab, properties tab, login tab, and register user tab. Also, there is one toolbar allowing the user to navigate through app and switch between these tabs.
    - The first tab, the application's main page, combines three components: a form component, a map component, and a table component.

        - The first component in the main tab is the form component that would be empty at the first opening. This component allows the user to look for the desired property by creating a query by filling the form. Using this form, a user could search properties based on their value, property type, size, and room numbers. The application turns these imported values into a query and sends them to the backend service. there is a predefined section for each query that user can feel any of them and look for the options under it

        - The second component is a map viewer that shows all the properties registered to the website and selected in the form component as dots.

        - The third component is the table component that shows all the properties selected in the form component. The table component shows the selected properties and their characteristics in a textual format in a table.

    - The second tab is the property tab, where users can see all the properties, browse between them, edit, add or remove them (if they are logged in to the website).

    - The third tab is the login tab, where registered users could log in to the system with their specific usernames and passwords.

    - The fourth tab is a form component allowing users to register themselves on the website providing required information.

- When the user opens the application for the first time, all the registered properties can be seen in both the map component (as dots) and table component (as raws)

- Different colors will show different types of properties (e.g., apartment, house) on the map.

- If it is not possible to show each property by dot, a combination of properties in an area will show by a number representing the number of properties in that area.

- Next phase is to make a query. When users navigate through the map (for example zooms in to a certain area, the system makes a request from the server and ask server to send back data from that certain area. This happens for the form component too. If a user fill one or many of the places in the query form, the system will automatically create a request based on the information in the form. Then checks if the query is valid and combine it with the spatial query that comes from the map section. Then send a final request to the server and ask for some specific properties with certain qualifications.

- In the backend, the system receives the request, creates a query, and asks for results from the database. The server will send a response with zero, one, or more property information based on the query. This response will be sent back to the front end and visualized in those three components.

- Every component visualizes the new data.

- If there is a mouse over from the user to one property, a new pop-up window will come up and give some additional information about that property.

- If the user clicks on one specific property, a new URL including that property webpage will be open in a new tab.

- The map component and the property component show the results from the form component and are constantly looking to this component for updates.

To provide the above functionalities, it seems that we can follow a client-server architecture for application development. The main benefit of such an architecture is its logical and physical separation of functionality and ease of implementation for junior developers. Of course, depending on the project needs and the level of development teams, different web architectures could be used for developing such an application. This architecture includes three main layers: the presentation layer (or user interface), the application's logic layer (business layer or backend), and the data storage layer.

The presentation layer, also called the Graphical User Interface (GUI), deals with serving static and dynamic contents to the user, loading web pages, and visualizations (e.g., maps, tables, and other visuals) in the application. This layer is also responsible for getting the user inputs. Html, CSS and JavaScript, and some JavaScript libraries and frameworks such as React.js, Vue.js, and Angular are normally used for developing this layer.

The application's logic layer (the backend) is mainly responsible for the application's business logic. This layer is responsible for saving and processing all the information from the presentation or data-storage layers. It simply processes and translates user requests into database queries on the one hand and converts query results from the database into the proper format and sends them to the presentation layer. This layer is often developed in programming languages such as Java, JavaScript, C#, and Python using popular frameworks such as STRUTS, ExpressJS, .Net, and Django.

The data storage layer (or database) has to store all necessary data about the application (in this case, the users and properties data) and includes both datasets and the database management system. Different types of technologies (relational databases such as PostgreSQL or non-relational databases such as MongoDB) could be used in this layer.

4.4. CIM development

This part is about developing CIM based on the application descriptions provided in the previous section and the instructions introduced in chapter three for developing CIM. As discussed earlier, this thesis uses the UML 2.0 activity diagram for CIM creation. This diagram is generated based on two sets of rules to construct UML activity diagrams discussed in the previous chapter.

Also, the following basic principles suggested by Rhazali et al. (Rhazali et al., 2018) for creating detailed activity diagrams are used. The rules of construction of a detailed model of the activity diagram are:

- to represent each of the model activities as a set of consecutive actions
- do not use this model for presenting a manual task that could not be done automatically
- represent the connections between activities
- an object node in the activity diagrams consists of an object state and one action output.

Based on the rules mentioned and the descriptions of the application and its mentioned viewpoint, the following activity model has been developed in Papyrus. The following diagram (figure 4-3) shows the CIM level model as a detailed activity diagram model (without swimlanes). In this model, it is tried to model each activity as several consecutive actions.

The step-by-step instructions on how to create an activity diagram is described in appendix D.

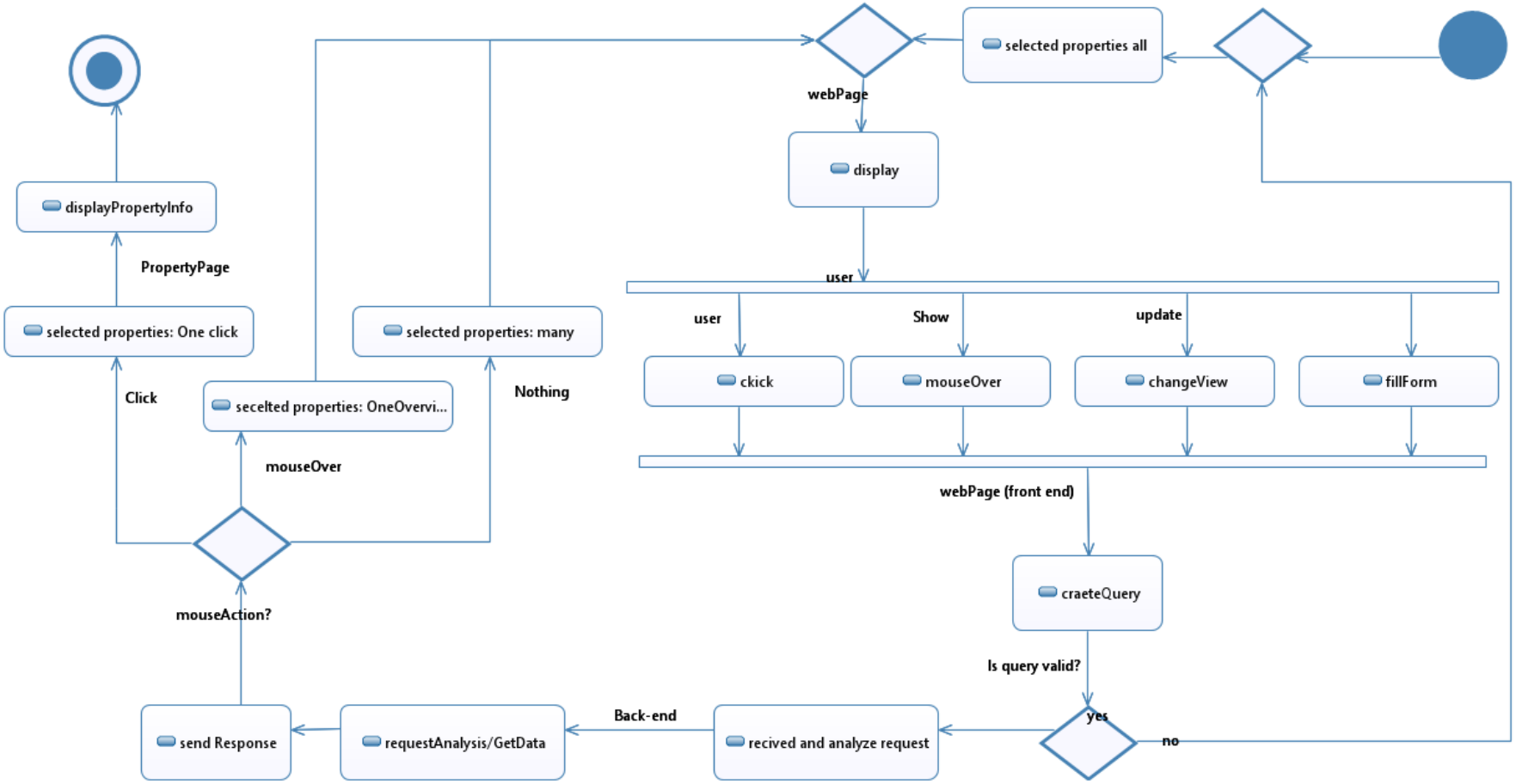


Figure 4-3- The application CIM

4.5. PIM development

The PIM is a model of information system regardless of the platform and technology in use. PIM is the result of CIM (business model, application logic) integrated with basic principles of designing information systems. It is independent from any particular technology and platform and could be used to produce several PSMs for different technologies and platforms.

After developing the PIM metamodel in chapter three, the PIM instance has to be generated from it since this model will be needed in the later stages of the work (e.g., for PIM to PSM model transformation). The PIM metamodel creates a framework to define classes, their attributes, and their methods for PIM based on the project specifications.

While creating a metamodel for a particular software project takes much time and needs constant care and consultation with domain experts, building a model based on this metamodel is more straightforward, does not need so much effort, and can be done relatively fast. After developing a metamodel for PIM, generating a PIM instance from it is relatively simple in EMF comparing to the metamodel development.

As described earlier, the EMF edit module is responsible for generating model instances of metamodels and their manipulation. All needed to be done is to identify the required classes for the project, their attributes and methods, and use the PIM metamodel to generate it. First, in EMF, a model generator has to be defined from a developed PIM metamodel. The model generator helps to create the model, edit, editor codes. It also creates the editor plugin, which provides wizards for creating new model instances. The editor plugin allows entering model information in those instances.

Here are the project specifications required in generating PIM from its metamodel and has to be imported in the editor plugin:

- The main application has three main elements: one database, one backend, and one front-end element.
- The backend element has two backend services (usersBack and properiesBack) each of them are connected to their respected data set.
- The application uses only one data base (name: mongoDB) which have only one data collection (name: Collection1). The data collection has two data sets (usersData and propertiesData)
- The application has one controller or user interface(frontend). The main user interface has one navigation bar and four different components (mainTab, usersTab, propetiesTab, and loginTab). Each of these components is connected to one backend service. One of these components (mainTab) has three different subcomponents (FORM, TABLE, and MAP), interacting with each other.

Figure 4-3 shows the generated PIM instance from the PIM metamodel based on the above specifications in the EMF environment. Appendix E provides step-by-step instructions on how to generate the PIM instance in EMF. Also, it is possible to find the resulted PIM in XMI format in appendix F.

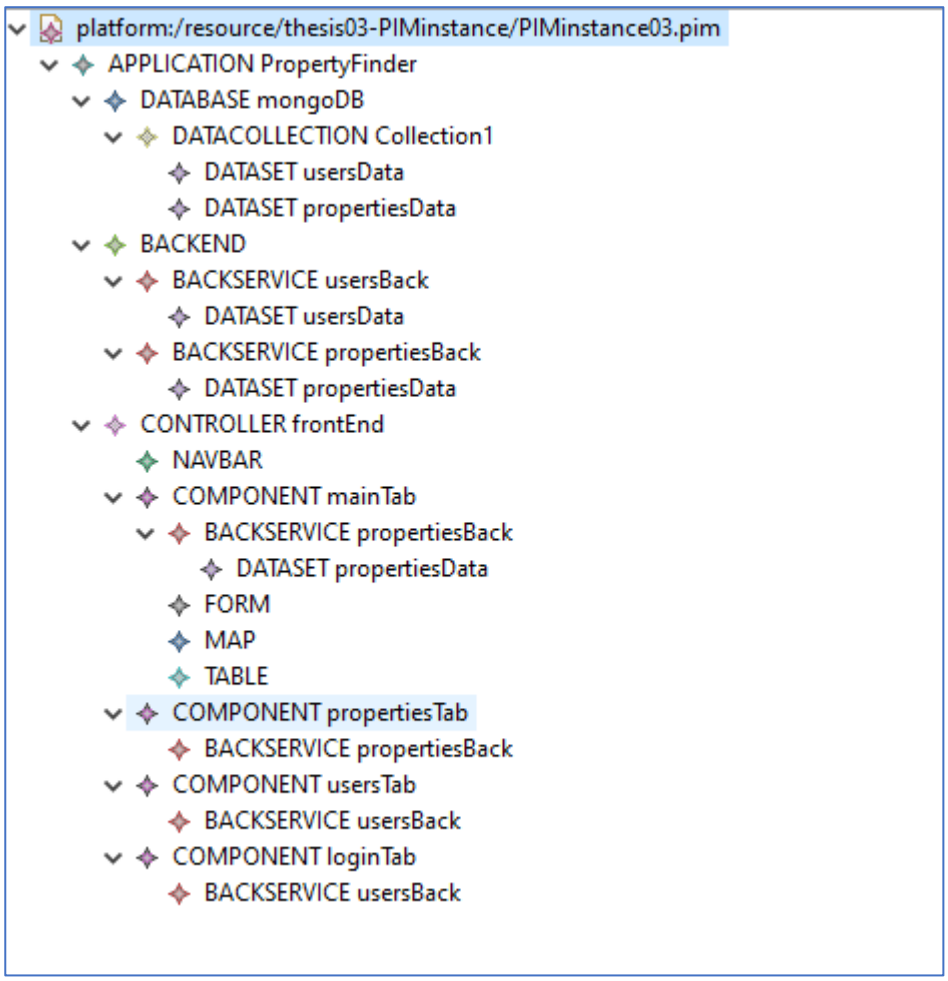


Figure 4-4- The PIM instance generated from PIM metamodel

The representation of the PIM instance as class diagram can be seen in figure 4-5. This diagram is another form of representation for the application's PIM. As you can see, different colors are assigned to classes based on their rule in the three layered structure of the application.

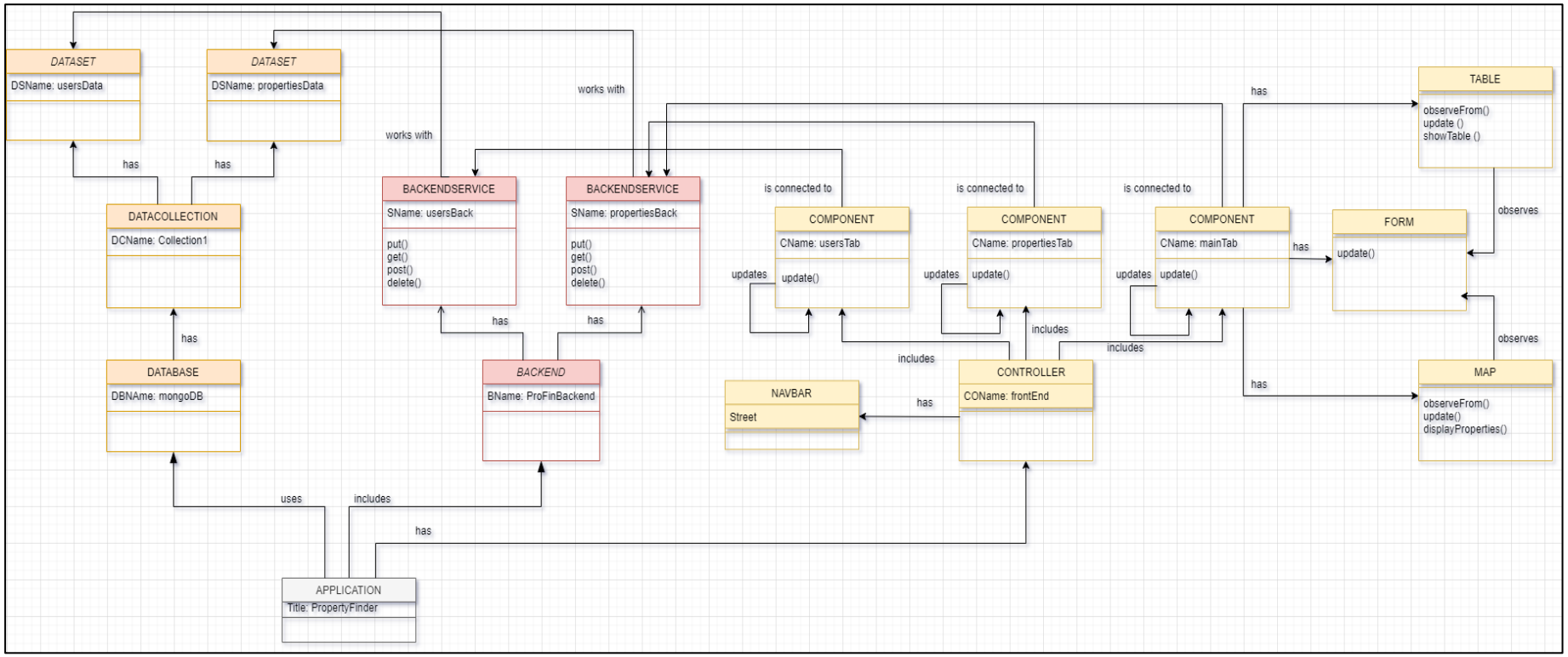


Figure 4-5- The PIM class diagram

4.6. PIM to PSM model transformation

This section deals with PIM to PSM transformation and describes the process in the QVT model transformation language. An overview of the mapping rules (pattern mappings and PIM to PSM metamodel mappings) is provided, and the process to generate a PSM instance using the transformation function has been explained. As described earlier, we use EMF for defining mapping rules and doing the model transformation (Barendrecht, 2010; Gerpheide, 2014; Gerpheide et al., 2016).

In the PIM to PSM transformation, mapping rules help describe how an object from an instance of PIM metamodel will be transformed into an instance of PSM metamodel. Mapping rules specify which objects, classes, attributes, and methods should be removed, added, or modified and how. Mapping rules and their definition are at the heart of MDA model transformation; they directly impact transformation function and, therefore, should be done by highly experienced developers with vast knowledge of software architecture and the target platform.

As described in the introduction, this chapter aims to show how model transformation based on design patterns could be used as a part of MDA software development and provides an overview of the process for an actual geospatial web application. So, there is no intention of going through a new modeling language (QVT operational) and explain its syntax. While we are going to explain the logic behind mapping rules and the transformation function, describing the whole language is outside of the scope of this thesis. There is too much technicality involved, and it does not serve the goal of this research. More information on QVT model transformation language, syntax, and how to use it could be found in Barendrecht 2010 and Gerpheide 2014.

As it was discussed in the previous chapter, the QVT model transformation language is a family of three languages: Core (QVTc), Relations (QVTr), and Operational Mappings (QVTo) (Barendrecht, 2010). For this thesis, we are going to use QVTo transformation language. According to the PBMDA approach, two sets of mapping rules are involved, one for the ordinary PIM to PSM model transformation and the other for pattern mappings. While this section is not going through the transformation language syntax, there will be a detailed explanation of the structure of transformation language, a pattern mapping (the Observer pattern), and one normal component mapping from PIM to PSM. A rather complete description of these transformation syntaxes in the QVTo transformation language can be found in Barendrecht, 2010.

The overall structure of transformation syntax in QVT operational is shown in figure 4-6. At first, the input and output models and corresponding metamodels have to be introduced to the system. Before starting programming in QVTo, the PIM and PSM metamodels have to be imported to the system separately. The first two lines (lines 2-3) are about introducing metamodels. In the transformation declaration (line 7), the input and output models (also known as source and target models) have to be identified. The purpose of the main function is to work as an entry point to the whole transformation, initialize the main variables, and introduce the first mapping. Mappings are the essential elements in QVTo transformation language as they specify how an input element turns to the output element. Each mapping in QVTo has three main parts: initialization, population, and end. The initialization section aims at variable initialization. The population section helps populate the output element based on the input and mapping input features to their equivalent output features. The end section describes the code to be executed after mapping is done. The "self" element in each mapping refers to the input element and provides access to one specific feature of the input element.

```
1  --Metamodel declaration (introducing metamodels)
2  modeltype PIM uses "http://pim/1.0.0";
3  modeltype PSM uses "http://psm/1.0.0";
4
5
6  --Transformation declaration
7  transformation PSMTransformation(in Source: PIM, out Target: PSM);
8
9  --The purpose of this function is to initialize variables and call the first mapping.
10 main() {
11
12     -- the mapping function has to be defined in the next step
13     Source.rootObjects()[APPLICATION] -> map AppToApplication();
14 }
15
16 mapping APPLICATION::AppToApplication(): application {
17     appName:= self.Title;
18     has:= self._uses -> map todatabase()
19     --runsBy:=
20     --isBasedOn:= self.includes -> map to
21 }
22
23 mapping PIM::DATABASE::todatabase() : PSM::database
24 when { self.DBName.startsWith("A"); }
25 {
26     name := self.DBName;
27     address:= 'dsfdf'
28
29 }
```

Figure 4-6- The main structure of QVTo

As described earlier, mapping rules for transforming some components from PIM to their corresponding components in PSM based on the Observer design pattern have been explained. The idea is to describe how these transformation rules work in the QVTo language. Figure 4-7 shows the identified observer pattern as problem specifications in the PIM metamodel and solution specifications in the PSM metamodel. In this section we are using mappings in QVTo to convert the solution specifications in PIM metamodel to the problem specifications in PSM metamodel. For this transformation, three classes in the PIM metamodel (MAP, TABLE, and FORM) must be mapped into three classes in the PSM metamodel (map, table, form). Figure 4-8 shows these mappings in QVTo transformation language syntax.

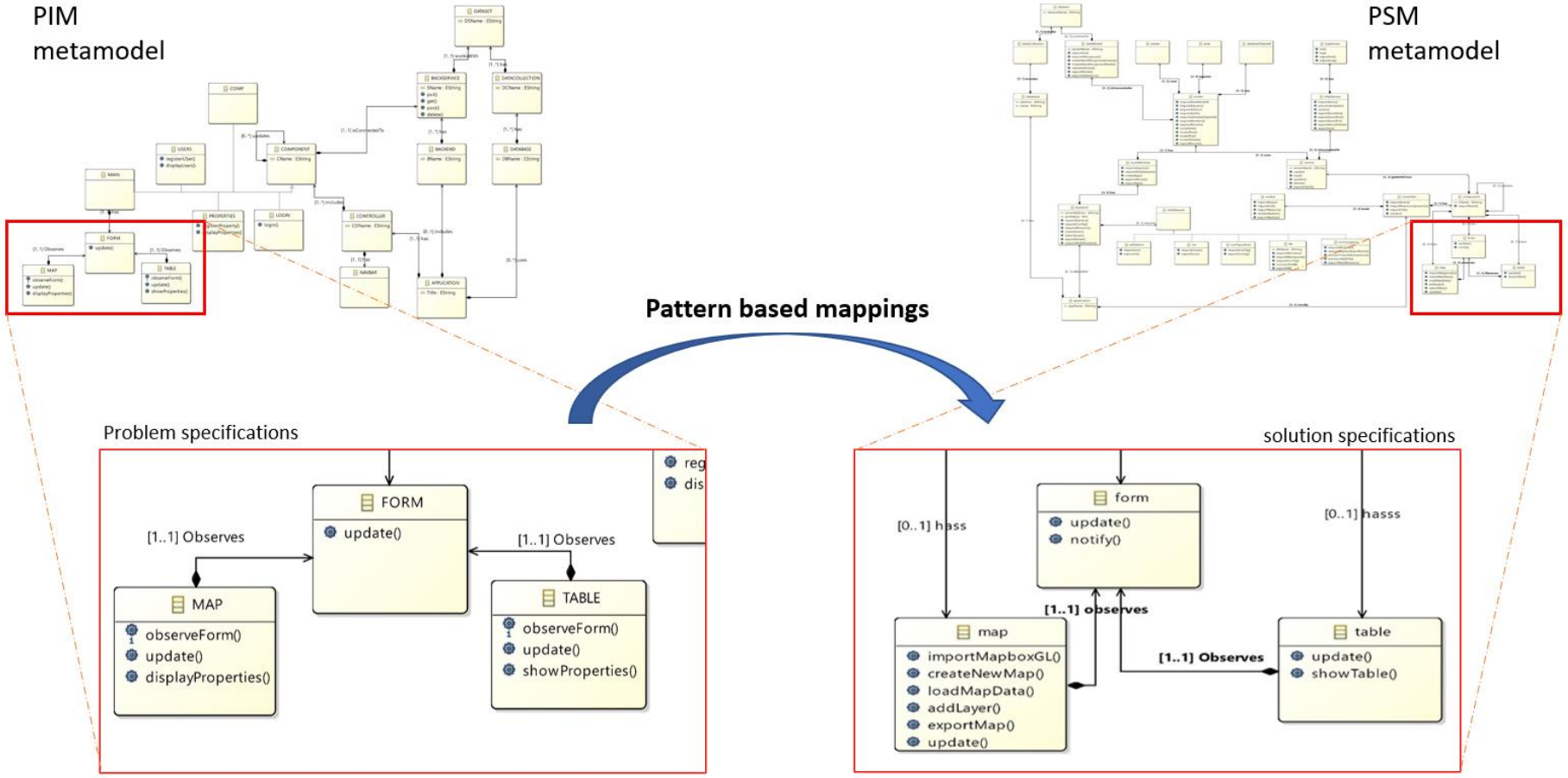


Figure 4-7- the observer pattern problem and solution specifications in the PIM and the PSM metamodels respectively

```
17 mapping APPLICATION::AppToApplication(): application {
18
19
20 -- declaration of mappings related to the observer design pattern
21
22 --mapping declaration, 'form' component
23 result.elements+=self.elements[FORM]->FORM_maptoSolutionSubject();
24
25 --mapping declaration, 'map' component
26 result.elements+=self.elements[MAP]->MAP_maptoSolutionObserver();
27
28 --mapping declaration, 'table' component
29 result.elements+=self.elements[TABLE]->TABLE_maptoSolutionObserver();
30
31 --mapping declaration, the 'observes' referebce
32 result.elements+=self.elements[observes]->maptoSolutionObserveAssociation()
33 }
34
35
36 --mapping: "form" component
37 mapping PIM::FORM::FORM_maptoSolutionSubject() : PSM:form
38 {
39     result.update := self.update.map PSMupdate();
40     result.notify:= object PSM::notify(name:= "Notify")
41 }
42
43 --mapping: "map" component
44 mapping PIM::MAP::MAP_maptoSolutionObserver() : PSM:map
45 {
46     result.update := self.update.map PSMupdate();
47     result.importMapboxGL := self.display.map importMapboxGL();
48     result.CreateNewMap := self.display.map CreateNewMap();
49     result.loadMapData := self.display.map loadMapData();
50     result.addLayer := self.display.map addLayer();
51     result.exportMap := self.display.map exportMap();
52
53 }
54
55 --mapping: the "table" component
56 mapping PIM::TABLE::TABLE_maptoSolutionObserver() : PSM:table
57 {
58     result.update := self.update.map PSMupdate();
59     result.showTable:= self.showProperties.map showTable();
60 }
61
62 --mapping: the "observes"
63 mapping PIM::observes::maptoSolutionObserveAssociation() : PSM:observes
64 {
65     result.name := self.name;
66     result.map := self.obs.map toSolutionObserver();
67     result.table := self.obs.map toSolutionObserver();
68     result.form := self.sub.map toSolutionSubject();
69 }
```

Figure 4-8- QVTo syntax - the observer pattern mappings

The first few lines (lines 20-33) are about mapping declarations of the pattern elements. There are declaration for four elements namely "FORM", "MAP", "TABLE", and the" observes" elements from the PIM metamodel. Each mapping declaration has a mapping operation. The rest of the syntax is about these mapping operations for each element and how the attributes, methods and properties of each element should be transformed from the PIM metamodel into the PSM metamodel.

By having the PIM and the PSM metamodels, the PIM instance and the model transformation function, we can generate the PSM instance automatically. The process is relatively simple in EMF and a proper step-by-step description can be found in this online tutorial (http://redpanda.nl/index.php?p=tutorial). The generated PSM instance could be found in the figure 4-9.

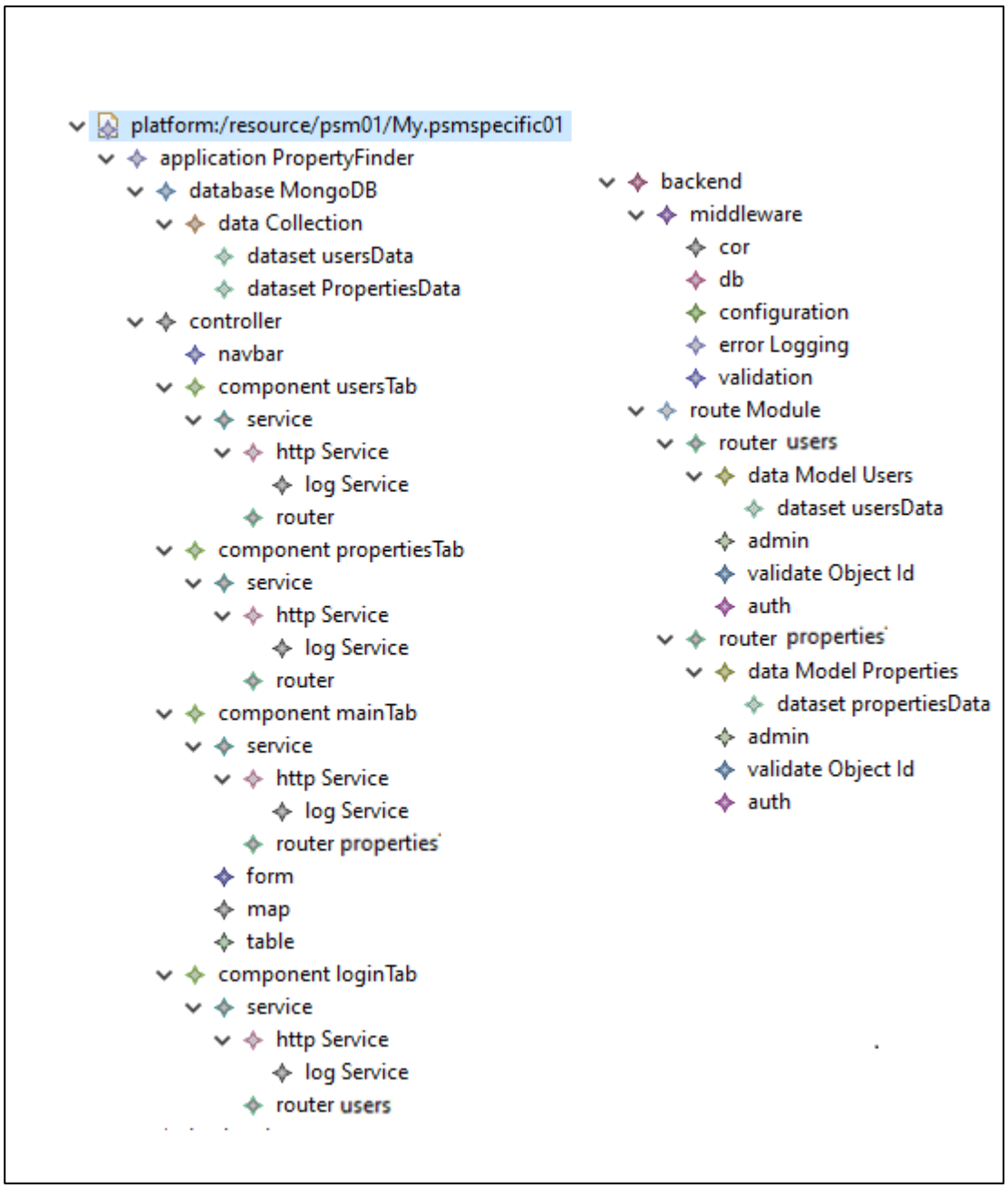


Figure 4-9- The generated PSM

Figure 4-10 shows the class diagram of the generated PSM. As it can be seen from the diagram, all necessary classes and methods are added based on the technology selected for the project implementation. Relevant classed and diagrams for a presentation layer based on React.js, required classes and methods for the back-end layer based on Node.js and all necessary classes and function for creating a MongoDB database.

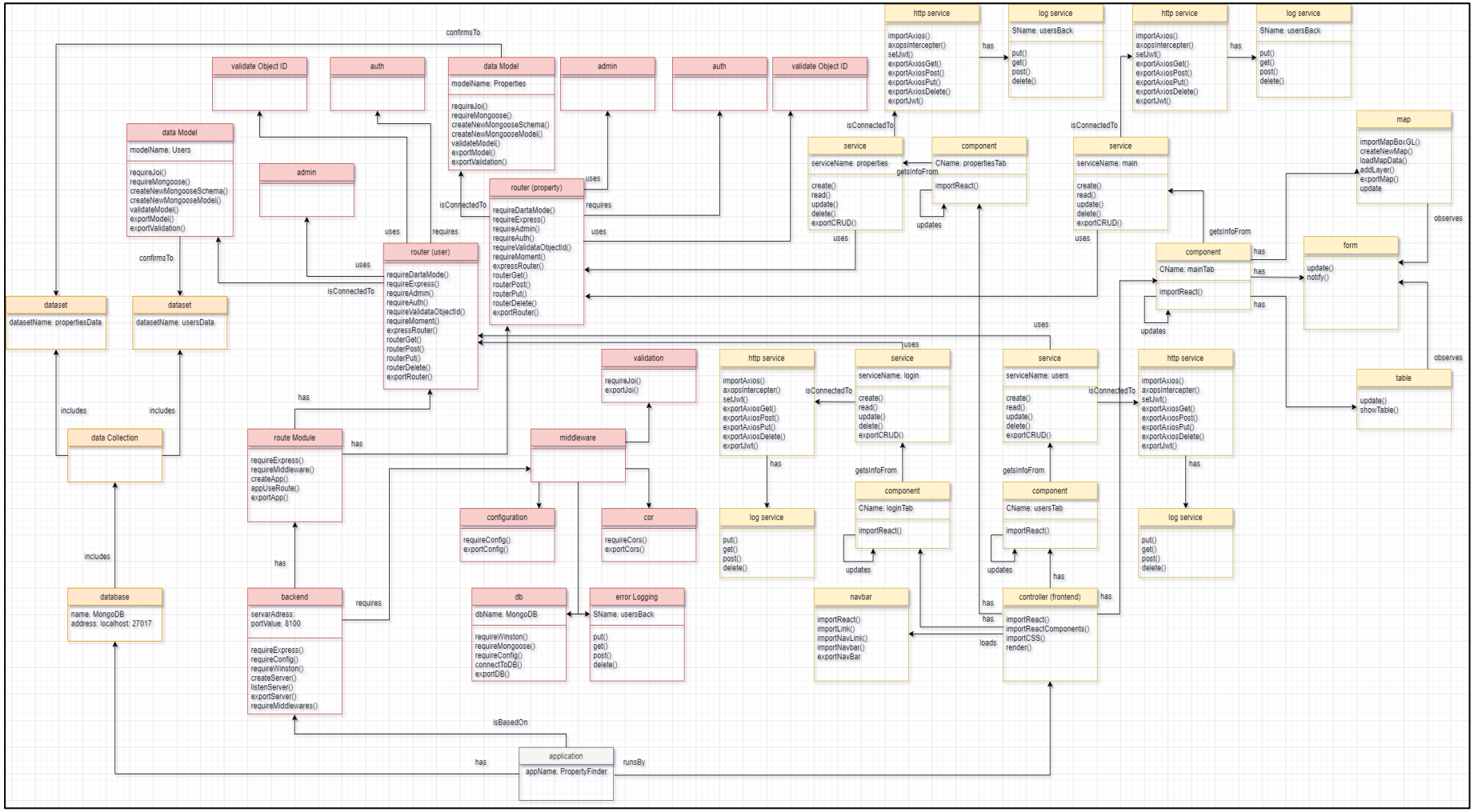


Figure 4-10- The PSM class diagram

4.7. Code generation

According to the MDA process, the next step after developing PSM is to automatically generate software code from it. Automatic code generation in MDA usually is done using different transformation languages such as MOF Model to Text Transformation Language. The automatic code generation is one of the highly technical steps in the whole MDA development process and as it has been mentioned before, the generated code is not enough for developing the whole application and needs significant enhancements. However, the quality of the generated code is directly related to the model to text transformation process and the tools for this transformation (e.g., the availability of software plugins) to automatically transform models into the code in a specific programming language.

Having the PSM there are tools in EMF that allow us to automatically generate source code in Java. But since the application is intended to be developed on JavaScript stack (MongoDb, ExpressJs, NodeJS, and ReactJS) and the fact that still there is not reliable transformation tool that works with JavaScript, the final source code for the application has been developed manually considering the generated code in Java with constant consultation with the application's PSM.

For implementing the model and creating the application, a dataset including 500 records of random coordinates, property types, room numbers, prices and other necessary values have been created and imported to a MongoDB database and the application code have been developed as described earlier. Figure 4-11 exhibits the application's main window (the main component) and its three sub-components (form component, map component, and table component).

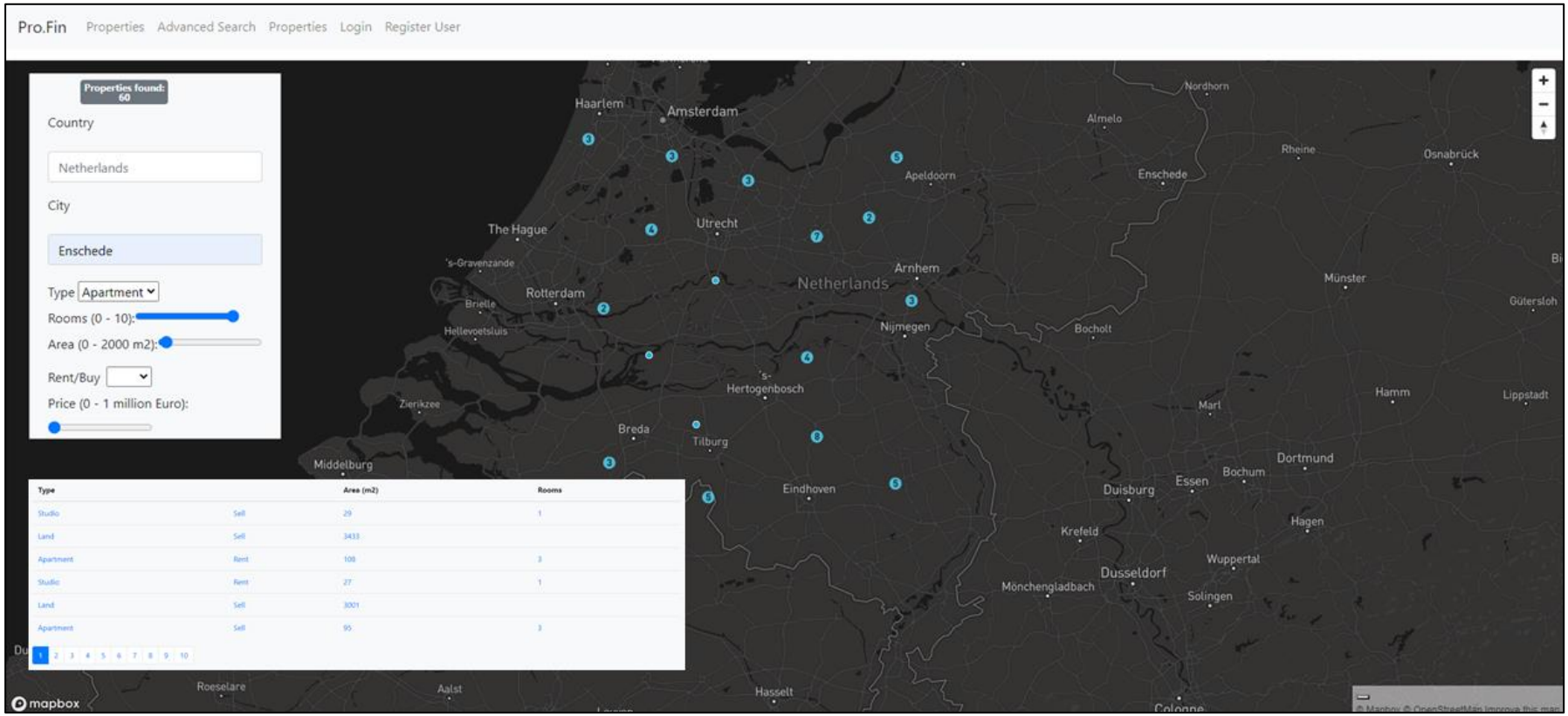


Figure 4-11- A screenshot of the developed application

5. DISCUSSION

In this research, we tried to introduce the Pattern Based Model Driven Architecture (PBMDA) to integrate software design patterns into the MDA methodology for web application development. We also provide a detailed description of how to develop geospatial web and mobile applications using this methodology. Using PBMDA for geospatial web application development, domain experts and developers have the opportunity to use the MDA's standard specifications for web application modeling along with common solutions for development challenges provided by design patterns. The applications developed based on PBMDA have several characteristics. They are loosely coupled with their data, implementing technologies and platforms. There is a separation of concerns between different stages of the development lifecycle. Also, using MDA and software design patterns in developing web applications provides the opportunity for better documentation and communication of the development process. All these lead to an increase in productivity, flexibility, and interoperability in application development.
Because of the importance of the OO paradigm and its rule as is the backbone for several new development methodologies and approaches in the software development field, such as software design patterns and the MDA, we discussed the concept of OO, its main principles and characteristics in the first step. Then we explained design patterns, their origins, and main characteristics, and how they could solve current problems and issues in the software development process. Then we explained web applications emphasizing geospatial web applications. We discussed their main features, their origins, and how they can be developed. Different methodologies for web application development have been reviewed, and the advantages and disadvantages of each methodology have been described. Finally, we started to explain the MDA process for software development, its basic principles, and its use for geospatial web application development.
After discussing the research basics, we proposed the PBMDA to integrate software design patterns into the MDA. We explained this methodology and introducing how it could improve the MDA process by incorporating the design pattern concept. We also describe how to create a platform-independent metamodel for a geospatial web application, a PSM metamodel for implementing this application into a set of JavaScript-based platforms using the PBMDA methodology. We have developed a geospatial web application for property finding using the PBMDA methodology and the created PIM and PSM to showcase the developed methodology. Then we have generated the code based on the developed PSM and develop an application from it.
Throughout this research we were guided by following research objectives:

- **Objective 1:** To investigate different web development methodologies
- **Objective 2:** Introducing MDA methodology for geospatial web development
- **Objective 3:** To propose a Pattern-Based Model Driven Architecture (PBMDA) for geospatial web application development by integrating software design patterns into MDA
- **Objective 4:** To develop a prototype geospatial web application based on the proposed pattern to demonstrating its applicability to the field

Questions and answers related to the first objective:
What are the essential criteria to consider for web application development?
Web applications have specific characteristics and requirements that should be considered in their development using a certain development methodology. These issues have been discussed in detail in chapter two. But here are the most important criteria that should be considered developing web applications:

- Comparing to conventional software, web applications have a shorter development lifecycle. That is why the development methodology for web applications should efficiently work through different phases of application development, from the analysis to design and implementation. This is the only way to ensure the final product will be ready on time for the market.
- The web development methodology should be able to create web applications that are loosely coupled from specific technologies and platform and data. There are constant changes and improvements in the web development landscape regarding development technologies and platforms, architectural design, and data forms. This makes it necessary to use development methodologies that enable web applications to be compatible with these new technologies and cope with changes.
- Web applications should deal with constantly changing user needs and requirements using specific web services and functionalities. These changes might require some updates in the application's logic or some of its objectives. Web development methodologies should consider this by providing the required facilities to easily upgrade applications based on changes in the application's logic and objectives.
- Developing web and mobile applications, it is crucial to consider security risk and privacy issues. These subject are gaining more attention as security challenges are among the most important risks that web applications are facing.
- It is of high importance to document the application development process since it makes it easier to communicate between team members, more efficient for debugging, and easier maintenance and update.
- Web applications should be able to use different web services
- Web applications should be able to perform well under different situations such as network traffic. They should be reliable as tools for users as more people do more work using them.

What are the specific requirements of geospatial web applications that should be considered in the development process?

Aside from the general features mentioned above about web applications, geospatial web applications have special characteristics and requirements that should be considered in their development process. Here they are the most important requirements that should be considered in geospatial web application development:

- The possibility to work with different data types (e.g., spatial and non-spatial) with different formats from different sources. They should be able to combine these data, process them and create valuable information.
- Emphasis on different types of data visualization such as maps, charts, diagrams, and tables.
- They have to be able to consider privacy issues regarding geospatial data and its distribution.
- To have to be able to deal with massive amounts of geospatial data from different sources e.g., sensor data, conventional databases, satellite image, and maps.
- Providing enough processing power for their users to process spatial data and information, to do spatial analysis, and to visualize them.
- The ability to Dealing with different web services and APIs for example to collecting spatial data

What are the main approaches and methodologies for web application development?
This question has been answered in length in chapter two. Speaking of web development, there are two sets of philosophies: traditional web development methodologies and those based on the OO paradigm. In this research, we introduced another classification that categorizes web development methods into five categories. It is really hard work to separate development methodologies because of the interrelationships between them and that some of the newer methodologies are developed based on, the older ones. Here are these five categories:

- Data-driven methodologies that focus on the application data and uses a data-driven structure for application development. Hera is a good examples of web development methodologies in this category.
- Hypertext-oriented methodologies which focuses on the hypertext dimension of the web applications. Well know methodologies in this category are HDM and W2000.
- Object Oriented methodologies are mainly based on the OO notation. UWE and OOHDM are two famous development methodology in this category.
- Software oriented methodologies try to develop web application based on the principles of software design and the OO notation. An example would be WAE.
- MDE based methodologies. These methodologies are based on the prominent use of models in all stages of development process such as MDA.

What is the best methodology for web application development?
An ideal methodology for web application development should be flexible to work with different web services and data sources, be adaptable to any platform development or technology upgrade, and be inclusive to consider the needs and requirements of all users and stakeholders in the web development process.

Among the several methodologies for web application development, MDE based web development methodologies seem good since they are based on the principles of the OO, and they use models and model transformation in the development process. As a result, they can provide some level of automation in the process. However, this methodology is not used widely by many in the industry because of some technological issues.

When we are talking about choosing the best methodology for web application development, we need to consider several factors such as project requirements, available tools, and stakeholder's requirements before making any decisions. We also need to consider the level of knowledge and experience of the development team. This way, we can choose a methodology that can satisfy our needs and ensure that the development team can implement it.

Questions and answers related to the second objective:
What is MDA, and what are its main characteristics?
Developed by OMG, MDA is a methodology for software development that is mainly based on MDD and the OO. MDA is a set of standards for creating models and metamodels for different phases of software development. The automatic transformation between models in different abstraction levels happens by model transformation, and the final transformation generates the application code from a model. There are three main models in the MDA process, namely CIM, PIM, and PSM. Using MDA for software and web development, we can have separation of concerns, some level of automation, and better documentation and communication between software development team members.

How does the MDA approach addresses those criteria important for geospatial web application development?

The main foundation of MDA is the architectural separation of concerns (OMG, 2014; Osis & Nazaruka, 2008) in which it separates a system's specifications from its implementation in a certain platform (Barbosa et al., 2013; Rossi et al., 2016). In the MDA development process, there are three models in three different abstraction level one independent from another: For example, PIM allows developers to focus on the system's main specifications (system analysis and design) without being concern about platform specific details (system implementation). This way, issues related to the implementation of information system on a particular platform could be addressed at another abstraction level by platform specialists and developers.

The separation of concerns facilitates traceability, reuse, and evolution in the information system. The system will be traceable because all steps related to its design, analysis, and implementation are developed and documented using models which makes it easier to understand and communicate. It also promotes reuse and evolution in the information system, because using model specifications and modelling concepts developers and domain experts are able to extend models and upgrade their model elements according to new requirements and reuse software elements in new specifications.

What are the main steps to develop a geospatial web application using the MDA approach?

Developing a web geospatial web application using MDA approach the following steps have to be taken:

1- Developing a DSL for GI domain

2- Developing the platform specific metamodel for the application based on the technical specifications (such as platform for application implementation)

3- Defining PIM to PSM transformation rules based on these metamodels and creating transformation function

4- Defining the application logic in the form of text

5- Creating CIM in the form of UML activity diagram using the application logic

6- Generating PIM from DSL and with the help of CIM

7- Developing PSM using the generated PIM and transformation function developed in item 3

8- Automatic code generation from PSM

9- Code revision and application development

If already a DSL is defined, we can skip the first three steps and start from step four.

What are the negative issues working with the MDA?

There are several (not necessarily negative) points that should be noticed when we trying to work with the MDA.

Several points (not necessarily negative) should be noticed when we try to work with the MDA.

- It should be noticed that this methodology is not fully automatic, and some manual enhancements might be necessary for different stages of application development with MDA (such as tuning models and metamodels, model transformations). However, this approach is still under development and improvement, and some of these issues related to the automation rate will be solved shortly.

- While the methodology generates the software code from PSM at the final stage, The automatically generated code covers almost 30% of the code needed for application development. This code has to be manipulated and enhanced manually to be helpful for the application.
- There are limitations in using UML for modelling problems and solutions in some particular domains. In this case DSLs have to be defined which can be a lengthy and costly process.
- For using this methodology for a particular domain on a specific platform, there is a need for highly knowledgeable developers and domain experts, and platform specialists to develop PIM and PSM metamodels. But once we created these metamodels for the domain and the particular platform, generating new applications would be easy and could be done by domain experts.

Questions and answers related to the third objective:

What are design patterns, and how they can improve the web development process?

As described in chapter 2, software design patterns are common solutions for recurrent problems in the software development process. During web development, there are many times that developers face problems with the same root and structure (in different contexts). These problems could be solved using standard solutions. Knowing this, developers could use these predefined solutions whenever they face a particular problem, and it could be solved by using a specific design pattern. Software design patterns can significantly impact the efficiency of web development projects since they improve the level of communication between developers, provide the opportunity for code and algorithm reuse, and facilitate documentation.

What are the main challenges in geospatial web application development using MDA?

Aside from some of the downsides of using the MDA for web application development that have been already discussed, there are still particular issues for using this methodology for geospatial web application development. One minor issue is that using structured methods and frameworks for geospatial applications (particularly the MDA methodology) is not widespread in the GI domain. Many developers and field experts are not aware of them or look at them as fancy approaches used for academic publication. However, the primary issue is initiating and developing a DSL for the GI domain (because of the UML limitations in capturing and expressing GI domain particular problems and solutions). Such DSL for GI domain barely exists, and it takes a lot of time and energy to develop one. But once the DSL for the GI domain is created, it would be highly convenient to create different geospatial applications. Also, limitations in current standards and platforms in application development, making it harder for developer and GI domain experts to incorporate the MDA-related concepts into their work.

How can software design patterns be integrated into MDA to create a Pattern-Based Model Driven Architecture (PBMDA)?

One way to integrate software design patterns into the MDA process is through PIM to PSM transformation as suggested by OMG (OMG, 2003) and others (Kim et al., 2017; Seffah, 2015). The idea is that when creating PIM and PSM metamodels, developers have to consider the possibility of applying design patterns in these two metamodels. This way, we can have two sets of model transformation rules to create a model transformation function—the pattern mapping rules and ordinary PIM to PSM transformation rules. As a result, when we import a PIM into the transformation function, the output would be a pattern-based PSM. The whole idea of defining metamodels based on design patterns and pattern-based model transformation has been described in Chapter 3.

Questions and answers related to the fourth objective:

What are the user requirements in a proposed geospatial web application?

Generally speaking, geospatial web application users should be able to get, edit, visualize, download, and upload geospatial data on the web. They should be able to access various types of data (geospatial and non-spatial, vector or raster, sensor data) from multiple sources and different web services and combine them and process them. Chapter 2 describes in more detail the actual needs and requirements of the geospatial users.

Speaking on the "property finder" application developed in chapter 4, application users need to have access to property data and see the list of all the properties, define queries from it and visualize the results in the form of maps, charts, and tables. They also should be able to register a property in the database or remove it from the database.

What are the main functionalities (components) in this application to address user requirements?

In this research, we tried to use the PBMDA methodology for developing a geospatial web application for locating properties (such as houses, apartments, and studios) for rent or sale. The application provides information about suitable properties to the user based on the user requirements by generating queries about the area, price, location, and other property features. The main designed functionalities of this application are the ability to store property data, define queries for the data, and visualize the query results in different formats (e.g., maps, charts, and tables). To provide these functionalities, we specifically used JavaScript libraries such as Mapbox GL JS to visualize data on the map (using WMS and WFS formats) and D3 (Data-Driven Documents) for visualizing data in the form of charts and diagrams.

How can we use the PBMDA approach to develop a web application for one of these functionalities?

To develop a geospatial web application using the PBMDA methodology and provide the desired user functionalities in this application, the first thing to do is create a PIM metamodel for this domain. Then based on generated metamodel and some additional information regarding the application implementation (e.g., technological requirements and desired platform to implement the application), a PSM metamodel has to be created. Transformation rules have to be defined for transforming PIM metamodel into PSM metamodel and considering the role of design patterns in these metamodels. By defining model transformation rules, we can create a model transformation function that generates an output PSM from an input PIM.

Then it is time to create CIM based on the application business model. Then PIM has to be generated from its metamodel and based on the defined CIM. Having PIM and model transformation function, PSM could be generated automatically. The last phase is to generate code from PSM for the application implementation.

5.1. Research limitations

- One of the significant limitations of this research was the lack of DSL (or the PIM metamodel) for the GI domain. Creating a DSL is one of the most complicated tasks in the whole PBMDA process.
- The developed DSL (PIM metamodel) in this research cannot be considered a perfect DSL for the GI domain. The purpose of this metamodel is to show the applicability of the research approach. Generating a DSL for the GI domain is not an easy task. It requires knowledgeable domain experts and software developers and might need some time to be completed by international organizations or research facilities.

- The lack of informative articles and detailed reports on using the MDA for web application development was another issue in this research. Unfortunately, most of the literature on this topic only covers one or two parts of the whole MDA process, for example, just model transformation. This way, it becomes hard to research the whole MDA process for web application development.

- Although we tried to explain the whole MDA methodology for developing geospatial web applications, we did not cover the last part of the process (automatic generation of application's code from the PSM and code edition) since it is a highly technical issue and is outside of the research scope.

5.2. Suggestions and Recommendations for future work

- While the application development and working with models and metamodels is not a task of GI domain experts and GIS professionals, they should have a minimum knowledge of these concepts to understand software models and work with software engineers and developers to develop better geospatial applications. Universities and other educational centers should offer elementary courses and lectures on computer modeling and metamodeling for students in the GI domain.

- Separate research on the automatic identification and use of software design patterns for geospatial web applications has to be conducted. In this research, we can work on the automatic detection of design patterns in MDA models. It is also necessary to know software design patterns in more detail and find out which design patterns could be used more in geospatial web applications.

- Separate research could be conducted for developing several Platform Specific metamodels for the same domain problem and transformation between these PSMs. This means by having one PIM, we can generate multiple PSMs and do the model transformation between them. For example, we can easily switch between different platforms for application implementation.

- We suggest that all centers and organizations working in the GI domain (such as OGC), universities, and research centers start developing DSLs for this domain. This way, we can have multiple DSLs, which could be used to generate new and more elaborate DSLs. By having DSLs, developing a geospatial web application using the MDA methodology would be simple and efficient for GI domain experts.

REFERENCES

- Adnan, M., Singleton, A., & Longley, P. (2010). Developing efficient web-based GIS applications. *(CASA Working Papers 153). Centre for Advanced Spatial Analysis (UCL): London, UK. (2010)* . http://www.casa.ucl.ac.uk/publications/workingPaperDetail.asp?ID=153
- Ahmed, E. A. (2013). *Getting Started with Model Driven Development and Domain Specific Modeling.*
- Alesheikh, A., Helali, H., & Behroz, H. (2002). Web GIS: Technologies and Its Applications. *Symposium on Geospatial Theory, Processing and Applications*.
- Armstrong, D. (2006). The quarks of object-oriented development. *Commun. ACM*, *49*, 123–128. https://doi.org/10.1145/1113034.1113040
- Avgeriou, P., & Zdun, U. (2005). Architectural Patterns Revisited - A Pattern Language. *EuroPLoP*.
- Aydinoğlu, A. Ç., & Kara, A. (2019). Modelling and publishing geographic data with model-driven and linked data approaches: case study of administrative units in Turkey. *Journal of Spatial Science*, *64*(1), 11–31. https://doi.org/10.1080/14498596.2017.1368420
- Barbosa, P., Contreras, C., & Murillo, J. (2013). MDA and Separation of Aspects: An approach based on multiple views and Subject Oriented Design. *AOSD'05*.
- Barendrecht, P. J. (2010). *Modeling transformations using QVT Operational Mappings*.
- Baresi, L., Garzotto, F., & Paolini, P. (2001). Extending UML for modeling web applications. *Proceedings of the Hawaii International Conference on System Sciences*, 89. https://doi.org/10.1109/HICSS.2001.926350
- Batty, M., Hudson-Smith, A., Milton, R., & Crooks, A. (2010). Map mashups, Web 2.0 and the GIS revolution. *Annals of GIS*, *16*, 1–13. https://doi.org/10.1080/19475681003700831
- Betari, O., Filali, S., Azzaoui, A., & Boubnad, M. (2018). Applying a model driven architecture approach: Transforming CIM to PIM using UML. *International Journal of Online Engineering (IJOE)*, *14*, 170. https://doi.org/10.3991/ijoe.v14i09.9137
- Borchers, J. O. (1999). *Pattern Languages in Human—Computer Interaction – Suite Overview*.
- Brambilla, M., Cabot, J., & Wimmer, M. (2012). Model-Driven Software Engineering in Practice. In *Synthesis Lectures on Software Engineering* (Vol. 1). https://doi.org/10.2200/S00441ED1V01Y201208SWE001
- Brambilla, M., Comai, S., Fraternali, P., & Matera, M. (2008). *Designing Web Applications with Webml and Webratio* (pp. 221–261). https://doi.org/10.1007/978-1-84628-923-1_9
- Briand, L. C., Labiche, Y., & Sauve, A. (2006). Guiding the Application of Design Patterns Based on UML Models. *2006 22nd IEEE International Conference on Software Maintenance*, 234–243. https://doi.org/10.1109/ICSM.2006.30
- Bruno, V., Tam, A., & Thom, J. (2005). Characteristics of Web applications that affect usability: A review. In *Proceedings of OZCHI 2005, Canberra, Australia*. https://doi.org/10.1145/1108368.1108445
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture - Volume 1: A System of Patterns*. Wiley Publishing.
- Cáceres, P., Marcos, E., & Vela, B. (2020). *A MDA-based approach for web information system development*.
- Capretz, L. F. (2003). A Brief History of the Object-Oriented Approach. *SIGSOFT Softw. Eng. Notes*, *28*(2), 6. https://doi.org/10.1145/638750.638778

- Ceri, S., Daniel, F., & Matera, M. (2004). *Extending WebML for modeling multi-channel context-aware Web applications*. https://doi.org/10.1109/WISEW.2003.1286806
- Chaffee, A. (2000). *What is a web application (or "webapp")?* http://www.jguru.com/faq/view.jsp?EID=129328
- Chan, F. K. Y., & Thong, J. Y. L. (2009). Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems*, *46*(4), 803–814. https://doi.org/10.1016/j.dss.2008.11.009
- Charatan, Q., & Safieddine, F. (2002). *A Review of Web Modelling Languages*.
- Chernichkin, A., & Nikiforova, O. (2009). An approach to classification of MDA tools. *J. Riga Technical University*, *38*, 72–83. https://doi.org/10.2478/v10143-009-0006-x
- Conallen, J. (2003). *Building Web Applications with UML*.
- Coplien, J. O. (1991). *Advanced C++ Programming Styles and Idioms*. Addison-Wesley.
- Czarnecki, K., Czarnecki, K., & Helsen, S. (2006). *Feature-based survey of model transformation approaches*. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.9674
- de Sousa Saraiva, J., & Rodrigues da Silva, A. (2008). Evaluation of MDE Tools from a Metamodeling Perspective. *Journal of Database Management*, *19*(4), 21–46. https://doi.org/10.4018/jdm.2008100102
- Deshpande, Y., & Hansen, S. (2001). Web Engineering:Creating a Discipline among Disciplines. *IEEE Multimedia*, *8*, 82–87. https://doi.org/10.1109/93.917974
- Deshpande, Y., Murugesan, S., Ginige, A., Hansen, S., Schwabe, D., Gaedke, M., & White, B. (2003). Web Engineering. *Journal of Web Engineering*, *1*.
- Dong, J., Zhao, Y., & Sun, Y. (2010). Design pattern evolutions in QVT. *Software Quality Journal*, *18*(2), 269–297. https://doi.org/10.1007/s11219-009-9093-8
- el Boussaidi, G., & Mili, H. (2012). Understanding design patterns—what is the problem? *Software: Practice and Experience*, *42*(12), 1495–1529.
- Favre, J.-M. (2004). *Foundations of Meta-Pyramids: Languages vs. Metamodels -- Episode II: Story of Thotus the Baboon1*.
- Finkelstein, A., Savigni, A., Kappel, G., Retschitzegger, W., Kimmerstorfer, E., Schwinger, W., Hofer, T., & Pröll, B. (2001). Ubiquitous Web Application Development - A Framework for Understanding. *Scientia Forestalis*.
- Fowler, M. (2006). *Writing Software Patterns*. https://www.martinfowler.com/articles/writingPatterns.html
- Fu, P., & Sun, J. (2010). *Web GIS: Principles and Applications* (Esri Press).
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns – Elements of Reusable Object—Oriented Software*. Addison—Wesley Longman, Inc.
- Garrigós, I., Gómez, J., & Cachero, C. (2003). Modelling dynamic personalization in web applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2722*, 472–475. https://doi.org/10.1007/3-540-45068-8_89
- Garrigós, I., Gomez, J., & Houben, G. J. (2010). Specification of personalization in web application design. *Information and Software Technology*, *52*(9), 991–1010. https://doi.org/10.1016/j.infsof.2010.04.001
- Garzotto, F., Mainetti, L., & Paolini, P. (1995). Hypermedia Design, Analysis, and Evaluation Issues. *Communications of the ACM*, *38*(8), 74–86. https://doi.org/10.1145/208344.208349
- Gerpheide, C. M. (2014). *Assessing and Improving Quality in QVTo Model Transformations*.
- Gerpheide, C. M., Schiffelers, R. R. H., & Serebrenik, A. (2016). Assessing and improving quality of QVTo model transformations. *Software Quality Journal*, *24*(3), 797–834.

- Gordillo, S., Balaguer, F., Mostaccio, C., & das Neves, F. (1999). Developing GIS applications with objects: A design patterns approach. *GeoInformatica*, *3*(1), 7–32. https://doi.org/10.1023/A:1009809511770
- Gorton, I. (2011). *Essential Software Architecture (2. ed.)*.
- Henderson-Sellers, B. (2005). UML - The good, the bad or the ugly? Perspectives from a panel of experts. *Software and Systems Modeling*, *4*(1), 4–13. https://doi.org/10.1007/s10270-004-0076-8
- Hojati, M. (2014, February 21). *What is is the Difference Between Web GIS and Internet GIS?* GIS Lounge.
- Houben, G.-J., Barna, P., Frasincar, F., & Vdovjak, R. (2003). *Hera: Development of Semantic Web Information*.
- Huang, Y. C., & Chu, C. P. (2014). Developing web applications based on model driven architecture. *International Journal of Software Engineering and Knowledge Engineering*, *24*(2), 163–182. https://doi.org/10.1142/S0218194014500077
- Jazayeri, M. (2007). *Some Trends in Web Application Development*. https://doi.org/10.1109/FOSE.2007.26
- Kardos, M., & Drozdova, M. (2010). Analytical method of CIM to PIM transformation in model driven architecture (MDA). *Journal of Information and Organizational Sciences*, *34*, 89–99.
- Kazato, H., Hayashi, S., Kobayashi, T., & Saeki, M. (2009). *Model-View-Controller Architecture Specific Model Transformation*.
- Kemp, K. (2008). Encyclopedia of Geographic Information Science. In *Encyclopedia of Geographic Information Science*. SAGE Publications, Inc. https://doi.org/10.4135/9781412953962
- Kherraf, S., Lefebvre, E., & Suryn, W. (2008). Transformation from CIM to PIM Using Patterns and Archetypes. In *Proceedings of the Australian Software Engineering Conference, ASWEC*. https://doi.org/10.1109/ASWEC.2008.4483222
- Kim, D. K., Lu, L., & Lee, B. (2017). Design pattern-based model transformation supported by QVT. *Journal of Systems and Software*, *125*, 289–308. https://doi.org/10.1016/j.jss.2016.12.019
- Kim, D.-K., & el Khawand, C. (2007). An approach to precisely specifying the problem domain of design patterns. *Journal of Visual Languages & Computing*, *18*(6), 560–591. https://doi.org/https://doi.org/10.1016/j.jvlc.2007.02.009
- Knapp, A., Koch, N., Moser, F., & Zhang, G. (2003). *ArgoUWE: A CASE tool for Web applications*.
- Koch, N., Kraus, A., & Munchen, L.-M.-U. (2002). *The Expressive Power of UML-based Web Engineering1*.
- Kriouile, A. (2015). An MDA Method for Automatic Transformation of Models from CIM to PIM. *American Journal of Software Engineering and Applications*, *4*(1), 1. https://doi.org/10.11648/j.ajsea.20150401.11
- Kruchten, P. (1999). What Is the Rational Unified Process? *The Rational Edge*.
- Kulkarni, V., & Reddy, S. (2003). Separation of Concerns in Model-Driven Development. *IEEE Softw.*, *20*, 64–69.
- Kuria, E., Kimani, S., & Mindila, A. (2019). A Framework for Web GIS Development: A Review. *International Journal of Computer Applications*, *178*, 6–10.
- Larman, C. (2004). *Applying UML and Patterns – An Introduction to Object—Oriented Analysis and Design and Iterative Development* (3rd ed.). Prentice Hall.
- Li, S., Dragicevic, S., Bert, V., & eds. (2011). *Advances in Web-based GIS, Mapping Services and Applications*.
- Li, S., Dragicevic, S., & Veenendaal, B. (Eds.). (2011). *Advances in Web-based GIS, Mapping Services and Applications* (1st ed.). CRC Press.
- Li, Y. F., Das, P. K., & Dowe, D. L. (2014). Two decades of Web application testing - A survey of recent advances. In *Information Systems* (Vol. 43, pp. 20–54). Elsevier Ltd. https://doi.org/10.1016/j.is.2014.02.001
- Liu, Y., & Wang, Y. (2011). *A study of metamodeling based on MDA*. https://doi.org/10.1109/ICCRD.2011.5764107

- Livshits, B. (2005). *Turning Eclipse Against Itself: Finding Bugs in Eclipse Code Using Lightweight Static Analysis*.
- Lowe, D. (2003). Web system requirements: an overview. *Requirements Engineering*, *8*(2), 102–113.
- Marcos, E., Vela, B., & Cáceres, P. (2003). A MDA-based approach for web information system development. *In Proceedings of Workshop in Software Model Engineering*. https://www.researchgate.net/publication/228847177
- Martínez-García, A., García-García, J. A., Escalona, M. J., & Parra-Calderón, C. L. (2015). Working with the HL7 metamodel in a Model Driven Engineering context. *Journal of Biomedical Informatics*, *57*, 415–424. https://doi.org/10.1016/j.jbi.2015.09.001
- *Mash-up | Definition of Mash-up by Merriam-Webster*. (n.d.). Retrieved July 5, 2021, from https://www.merriam-webster.com/dictionary/mash-up
- McArthur, K. (2008). *Pro PHP: Patterns, Frameworks, Testing and More*. Apress.
- Mellor, S. J., Clark, A. N., & Futagami, T. (2003). Model-driven development - Guest editor's introduction. *IEEE Software*, *20*(5), 14–18. https://doi.org/10.1109/MS.2003.1231145
- Meservy, T., & Fenstermacher, K. (2005). Transforming software development: An MDA road map. *Computer*, *38*, 52–58. https://doi.org/10.1109/MC.2005.316
- Molina-Ríos, J., & Pedreira-Souto, N. (2020). Comparison of development methodologies in web applications. In *Information and Software Technology* (Vol. 119, p. 106238). Elsevier B.V. https://doi.org/10.1016/j.infsof.2019.106238
- Moreno, N., Romero, J. R., & Vallecillo, A. (2007). An Overview Of Model-Driven Web Engineering and the Mda. In *Web Engineering: Modelling and Implementing Web Applications* (pp. 353–382). Springer London. https://doi.org/10.1007/978-1-84628-923-1_12
- Neumann, A. (2008). Web Mapping and Web Cartography. In S. Shekhar & H. Xiong (Eds.), *Encyclopedia of GIS* (pp. 1261–1269). Springer US. https://doi.org/10.1007/978-0-387-35973-1_1485
- Neumann, A. (2012). Web mapping and web cartography. In *Springer Handbook of Geographic Information* (pp. 567–587). https://doi.org/10.1007/978-3-540-72680-7_14
- Nguyen, V.-C., & Richta, K. (2014). *Domain Specific Language Approach on Model-driven Development of Web Services*.
- Nolte, S. (2010). *QVT - Operational Mappings*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-92293-3
- Object Management Group. (2017). *Unified Modeling Language (UML), Version: 2.5.1*.
- OMG. (2003). *MDA Guide Version 1.0.1*.
- OMG. (2011). *Meta Object Facility (MOF) Core Specification*.
- OMG. (2014). *Model Driven Architecture (MDA) Guide rev. 2.0*.
- Open Geospatial Consortium (OGC). (2021, January). *Glossary of Terms*. ogc.org/ogc/glossary/w
- Osis, J., & Nazaruka, E. (2008). *Enterprise Modeling for Information System Development within MDA*. https://doi.org/10.1109/HICSS.2008.150
- Pang, X., Ma, K., & Yang, B. (2011). Design pattern modeling and implementation based on MDA. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6988 LNCS*(PART 2), 11–18. https://doi.org/10.1007/978-3-642-23982-3_2
- Plessers, P., Casteleyn, S., & Troyer, O. (2005). *Semantic Web development with WSDM. 185*.
- Prakash, A., & Karri, N. (2018). Application of Design Patterns for Designing GIS Map Display Component. *Journal of Remote Sensing & GIS*, *07*. https://doi.org/10.4172/2469-4134.1000248

- Rahmouni, M., & Mbarki, S. (2011). MDA-based ATL transformation to generate MVC 2 web models. *International Journal of Computer Science & Information Technology*, *3*. https://doi.org/10.5121/ijcsit.2011.3405
- Rhazali, Y., Hadi, Y., Chana, I., Lahmer, M., & ab, R. (2018). A model transformation in model driven architecture from business model to web model. *IAENG International Journal of Computer Science*, *45*, 104–117.
- Riehle, D. (2011). Lessons learned from using design patterns in industry projects. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *6510*, 1–15. https://doi.org/10.1007/978-3-642-19432-0_1
- Riehle, D., & Züllighoven, H. (1996). Understanding and Using Patterns in Software Development. *Theory Pract. Object Syst.*, *2*, 3–13.
- Rode, G. (2008). *Evaluating Software Design Patterns — the "Gang of Four" patterns implemented in Java 6*.
- Rode, J., Rosson, M. B., & Pérez-Quiñones, M. A. (2005). *The Challenges of Web Engineering and Requirements for Better Tool Support*. Department of Computer Science, Virginia Polytechnic Institute & State University. https://vtechworks.lib.vt.edu/handle/10919/20125
- Rodriguez, A., Guzmán, I. G.-R. de, Fernández-Medina, E., & Piattini, M. (2010). Semi-Formal Transformation of Secure Business Processes into Analysis Class and Use Case Models: An MDA Approach. *Inf. Softw. Technol.*, *52*(9), 945–971. https://doi.org/10.1016/j.infsof.2010.03.015
- Rossi, G., Schwabe, D., & Guimarães, R. (2001). *Designing personalized web applications*. https://doi.org/10.1145/371920.372069
- Rossi, G., Urbieta, M., Distante, D., Rivero, J. M., & Firmenich, S. (2016). 25 Years of Model-Driven Web Engineering: What we achieved, What is missing. *CLEI Electronic Journal*, *19*(1), 5–57. https://doi.org/10.19153/cleiej.19.3.1
- Saraiva, J. (2013). *Development of CMS-based Web Applications with a Multi-Language Model-Driven Approach*.
- Saraiva, J. d. S., & Silva, A. R. d. (2009). CMS-Based Web-Application Development Using Model-Driven Languages. *2009 Fourth International Conference on Software Engineering Advances*, 21–26. https://doi.org/10.1109/ICSEA.2009.12
- Schmidt, D. C. (2006). Model-driven engineering. In *Computer* (Vol. 39, Issue 2, pp. 25–31). IEEE Computer Society. https://doi.org/10.1109/MC.2006.58
- Schwabe, D., & Rossi, G. (1995). The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, *38*(8), 45–46. https://doi.org/10.1145/208344.208354
- Schwinger, W., & Koch, N. (2006). *Modeling Web Applications*.
- Seffah, A. (2015). *POMA: Pattern-Oriented and Model-Driven Architecture* (pp. 155–180). https://doi.org/10.1007/978-3-319-15687-3_8
- Sharifi, H. R., & Mohsenzadeh, M. (2012). A New Method for Generating CIM Using Business and Requirement Models. *World of Computer Science and Information Technology Journal*, *2*.
- Siau, K., Cao, Q., Siau, K., & Cao, Q. (2001). Unified Modeling Language: A Complexity Analysis. *Journal of Database Management (JDM)*, *12*(1), 26–34. https://EconPapers.repec.org/RePEc:igg:jdm000:v:12:y:2001:i:1:p:26-34
- Soley, R., & OMG Staff Strategy Group. (2000). *Model Driven Architecture*.
- Sparx Systems. (2019). *Enterprise Architect 15 Reviewer's Guide*.
- Standing, C. (2002). Methodologies for developing Web applications. *Information and Software Technology*, *44*(3), 151–159. https://doi.org/10.1016/S0950-5849(02)00002-2
- Suh, Woojong. (2005). *Web engineering : principles and techniques*. Idea Group Pub.

- Taleb, M., Seffah, A., & Abran, A. (2007). Model-driven architecture for Web applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4550 LNCS*(PART 1), 1198–1205. https://doi.org/10.1007/978-3-540-73105-4_130

- Techopedia. (2021, March 12). *Web Mapping*. Techopedia. https://www.techopedia.com/definition/15584/web-mapping

- Tekavec, J., & Lisec, A. (2020). Cadastral data as a source for 3D indoor modelling. *Land Use Policy*, *98*, 104322. https://doi.org/10.1016/j.landusepol.2019.104322

- The Object Management Group. (2006). *Meta Object Facility (MOF) Core Specification*.

- The Object Management Group. (2007). *Meta Object Facility (MOF) 2.0 Query/View/- Transformation Specification*.

- 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

- Torres, V., Giner, P., & Pelechano, V. (2012). Developing BP-driven web applications through the use of MDE techniques. *Software and Systems Modeling*, *11*(4), 609–631. https://doi.org/10.1007/s10270-010-0177-5

- Veenendaal, B., Brovelli, M. A., & Li, S. (2017). Review of Web Mapping: Eras, Trends and Directions. *ISPRS International Journal of Geo-Information*, *6*(10), 317. https://doi.org/10.3390/ijgi6100317

- Vlissides, J. (1997). Patterns: The top 10 misconceptions. *Object Magazine*, *7*, 30–33.

- Wikipedia. (2021). *Web Mapping*. https://en.wikipedia.org/wiki/Web_mapping

- Wimmer, M., Schauerhuber, A., Schwinger, W., & Kargl, H. (2007). On the integration of web modeling languages: Preliminary results and future challenges. *CEUR Workshop Proceedings*, *261*.

- Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1990). *Designing Object-Oriented Software*. Prentice-Hall, Inc.

- Xiao-wei, & Xue, Y. (2011). *A Survey on Web Application Security*.

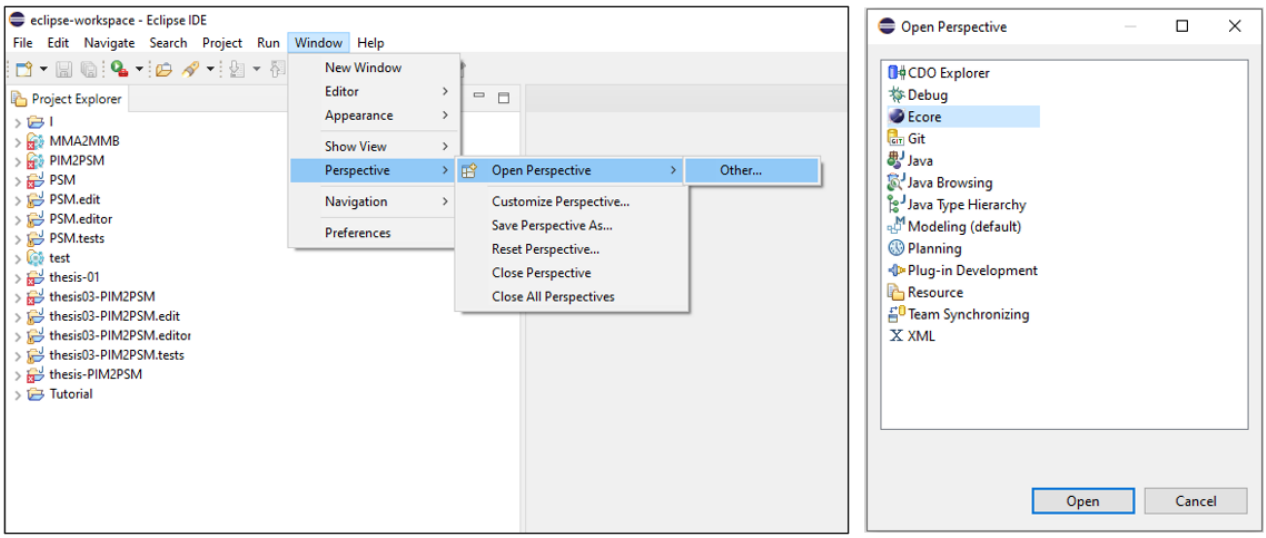
- Zadahmad Jafarlou, M., Moeini, A., & YousefzadehFard, P. (2010). New process: Pattern-based Model Driven Architecture. *Procedia Computer Science Jornal - World Conference on Information Technology by ELSEVIER*. https://doi.org/10.1016/j.protcy.2012.02.095

APPENDIXES

Appendix A: Creating a modeling project and a metamodel in EMF

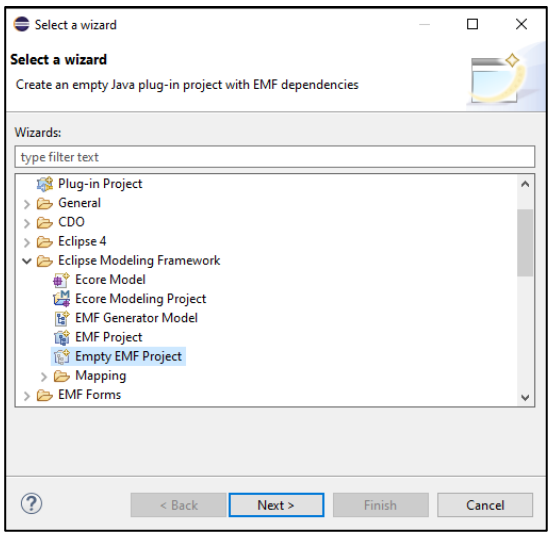
- In the Eclipse IDE, open the "Ecore" perspective.

Window → perspective → open perspective → ecore



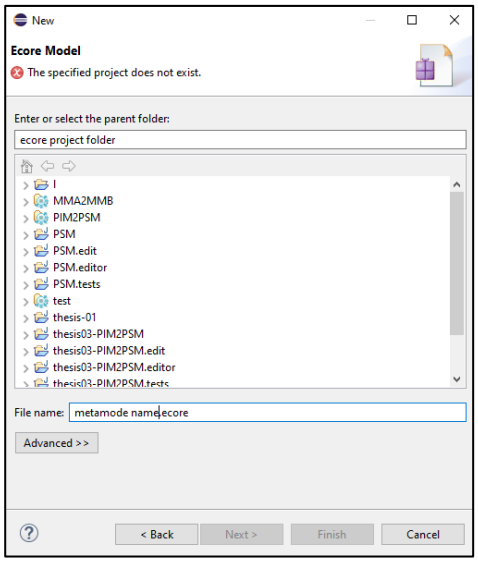
- Create an empty EMF project.

File → new → other… → Eclipse Modeling Framework → Empty EMF project

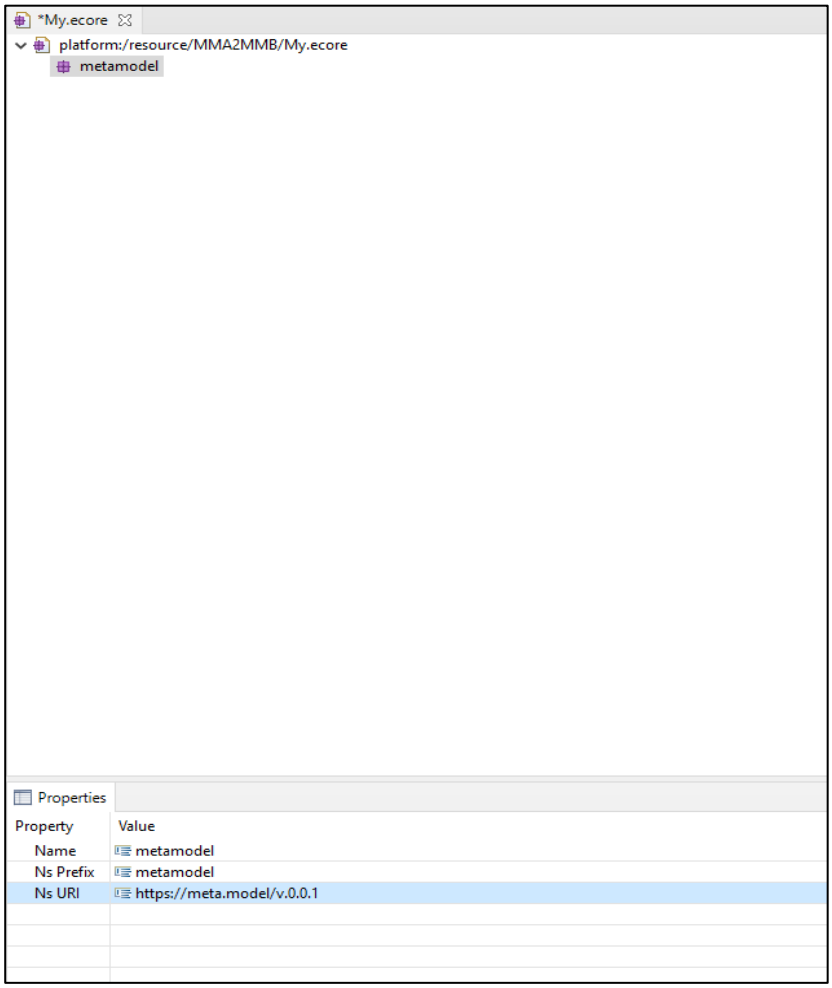


- Create a new Ecore metamodel

File → new → other … → Eclipse Modeling Framework → Ecore mode (then select the ecore modelling project and enter a name for the ecore metamodel)

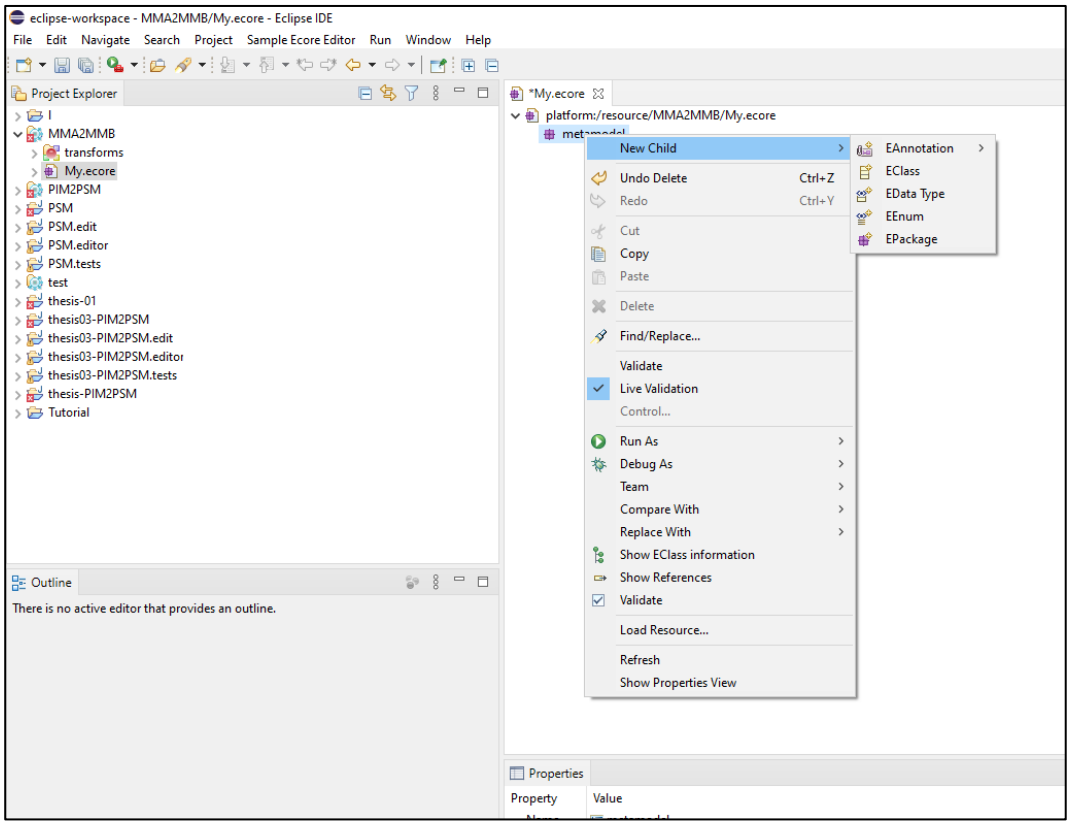


- Open the created ecore metamodel, enter values for “Name”, “Ns Prefix”, and “NS URI”.



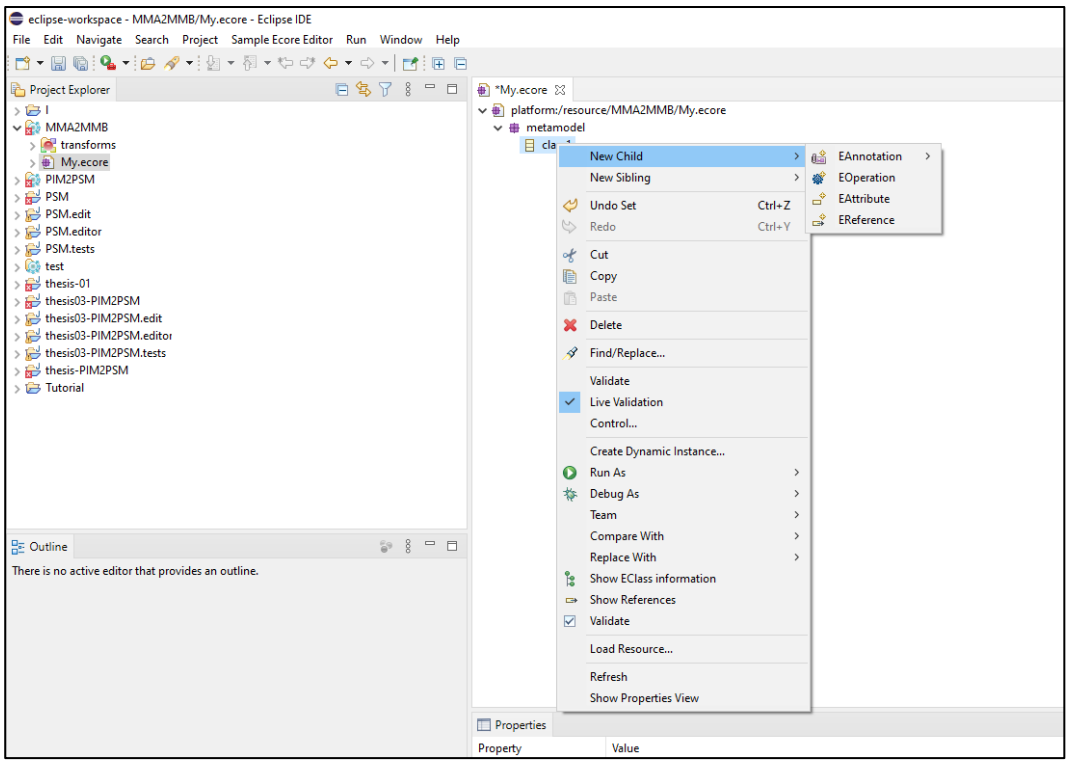
- Create classes, data types, enumerations and packages in your metamodel and add attributes, references and types to them.

Right click on the metamodel → new child



- Create annotations, attributes, references and operations for your classes.

Right click on the class in metamodel → new child



Appendix B: The Platform Independent Metamodel in XML format

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="PIM" nsURI="http://pim/1.0.0"
nsPrefix="pim">
  <eClassifiers xsi:type="ecore:EClass" name="APPLICATION">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Title" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="uses" upperBound="-1" eType="#//DATA-
BASE"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="includes" eType="#//BACKEND"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" eType="#//CONTROL-
LER"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DATABASE">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="DBName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" upperBound="-1"
        eType="#//DATACOLLECTION" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DATACOLLECTION">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="DCName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" upperBound="-1"
        eType="#//DATASET" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="DATASET">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="DSName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BACKEND">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="BName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" upperBound="-1"
        eType="#//BACKSERVICE" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BACKSERVICE">
    <eOperations name="put"/>
    <eOperations name="get"/>
    <eOperations name="post"/>
    <eOperations name="delete"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="worksWith" lowerBound="1"
        eType="#//DATASET" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="SName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="CONTROLLER">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="COName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" eType="#//NAVBAR"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="includes" lowerBound="1"
        upperBound="-1" eType="#//COMPONENT" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="COMPONENT">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="CName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="updates" upperBound="-1"
        eType="#//COMPONENT" containment="true"/>
```

```
    <eStructuralFeatures xsi:type="ecore:EReference" name="isConnectedTo" lowerBound="1"
        eType="#//BACKSERVICE" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" eType="#//FORM" contain-
ment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hass" eType="#//MAP" contain-
ment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasss" eType="#//TABLE"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="NAVBAR"/>
  <eClassifiers xsi:type="ecore:EClass" name="MAP">
    <eOperations name="observeForm" lowerBound="1"/>
    <eOperations name="update"/>
    <eOperations name="displayProperties"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Observes" lowerBound="1"
        eType="#//FORM" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="FORM">
    <eOperations name="update"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="TABLE">
    <eOperations name="observeForm" lowerBound="1"/>
    <eOperations name="update"/>
    <eOperations name="showTable"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Observes" lowerBound="1"
        eType="#//FORM" containment="true"/>
  </eClassifiers>
</ecore:EPackage>
```

Appendix C: The Platform Specific Metamodel in XML format

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="specific01" nsURI="http://spe-
cific.com/1.0.0" nsPrefix="specific-01">
  <eClassifiers xsi:type="ecore:EClass" name="application">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="appName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" upperBound="-1" eType="#//data-
base"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="runsBy" lowerBound="1"
        eType="#//controller" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="isBasedOn" eType="#//backend"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="backend">
    <eOperations name="requireExpress"/>
    <eOperations name="requireConfig"/>
    <eOperations name="requireWinston"/>
    <eOperations name="createSever"/>
    <eOperations name="listenServer"/>
    <eOperations name="exportServer"/>
    <eOperations name="requireMiddlewares"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="serverAddress" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="portValue" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="requires" eType="#//middleware"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" eType="#//route-
Module"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="controller">
    <eOperations name="importReact"/>
    <eOperations name="importReactcomponents"/>
    <eOperations name="importCSS"/>
    <eOperations name="render"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Loads" lowerBound="1"
eType="#//navbar"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" upperBound="-1"
        eType="#//component" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="database">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="address" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="includes" lowerBound="1"
        upperBound="-1" eType="#//dataCollection" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="dataCollection">
    <eStructuralFeatures xsi:type="ecore:EReference" name="includes" lowerBound="1"
        upperBound="-1" eType="#//dataset" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="dataset">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="datasetName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
```

```
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="middleware"/>
  <eClassifiers xsi:type="ecore:EClass" name="cor" eSuperTypes="#//middleware">
    <eOperations name="requireCores"/>
    <eOperations name="exportCors"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="db" eSuperTypes="#//middleware">
    <eOperations name="requireWinston"/>
    <eOperations name="requireMongoose"/>
    <eOperations name="requireConfig"/>
    <eOperations name="connectToDB"/>
    <eOperations name="exportDB"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="dbName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="configuration" eSuperTypes="#//middleware">
    <eOperations name="requireConfig"/>
    <eOperations name="exportConfig"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="validation" eSuperTypes="#//middleware">
    <eOperations name="requireJoi"/>
    <eOperations name="exportJoi"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="routeModule">
    <eOperations name="requireExpress"/>
    <eOperations name="requireMiddleware"/>
    <eOperations name="createApp"/>
    <eOperations name="appUseRoute"/>
    <eOperations name="exportApp"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" upperBound="-1"
        eType="#//router" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="router">
    <eOperations name="requireDataModel"/>
    <eOperations name="requireExpress"/>
    <eOperations name="requireAdmin"/>
    <eOperations name="requireAuth"/>
    <eOperations name="requireValidateObjectId"/>
    <eOperations name="requireMoment"/>
    <eOperations name="expressRouter"/>
    <eOperations name="routerGet"/>
    <eOperations name="routerPost"/>
    <eOperations name="routerPut"/>
    <eOperations name="routerDelete"/>
    <eOperations name="exportRouter"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="isConnectedto" lowerBound="1"
        eType="#//dataModel" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="uses" lowerBound="1" eType="#//admin"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="use" lowerBound="1" eType="#//vali-
dateObjectId"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="requires" lowerBound="1"
        eType="#//auth" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="navbar">
    <eOperations name="importReact"/>
    <eOperations name="importLink"/>
    <eOperations name="importNavLink"/>
    <eOperations name="renderNavbar"/>
    <eOperations name="exportNavbar"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="component">
    <eOperations name="importReact"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="getsInfoFrom" lowerBound="1"
```

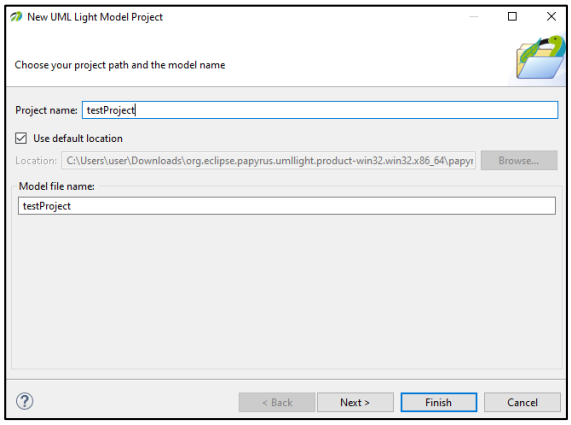
```
        eType="#//service" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="updates" eType="#//component"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="CName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" eType="#//form" contain-
ment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hass" eType="#//map" contain-
ment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasss" eType="#//table"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="map">
    <eOperations name="importMapboxGL"/>
    <eOperations name="createNewMap"/>
    <eOperations name="loadMapData"/>
    <eOperations name="addLayer"/>
    <eOperations name="exportMap"/>
    <eOperations name="update"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="observes" lowerBound="1"
        eType="#//form" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="form">
    <eOperations name="update"/>
    <eOperations name="notify"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="table">
    <eOperations name="update"/>
    <eOperations name="showTable"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="Observes" lowerBound="1"
        eType="#//form" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="service">
    <eOperations name="create"/>
    <eOperations name="read"/>
    <eOperations name="update"/>
    <eOperations name="delete"/>
    <eOperations name="exportCRUD"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="serviceName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="isConnectedTo" lowerBound="1"
        eType="#//httpService" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="uses" lowerBound="1" eType="#//router"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="httpService">
    <eOperations name="importAxios"/>
    <eOperations name="axiosIntercepter"/>
    <eOperations name="setJwt"/>
    <eOperations name="exportAxiosGet"/>
    <eOperations name="exportAxiosPost"/>
    <eOperations name="exportAxiosPut"/>
    <eOperations name="exportAxiosDelete"/>
    <eOperations name="exportJwt"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="has" lowerBound="1" eType="#//logS-
ervice"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="logService">
    <eOperations name="init"/>
    <eOperations name="log"/>
    <eOperations name="exportInit"/>
    <eOperations name="exportLog"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="errorLogging" eSuperTypes="#//middleware">
```

```
    <eOperations name="requireWinston"/>
    <eOperations name="requireExpressAsyncErrors"/>
    <eOperations name="winstonHandleExceptions"/>
    <eOperations name="winstonAddFile"/>
    <eOperations name="exportNewWinston"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="admin"/>
  <eClassifiers xsi:type="ecore:EClass" name="auth"/>
  <eClassifiers xsi:type="ecore:EClass" name="validateObjectId"/>
  <eClassifiers xsi:type="ecore:EClass" name="dataModel">
    <eOperations name="requireJoi"/>
    <eOperations name="requireMongoose"/>
    <eOperations name="createNewMongooseSchema"/>
    <eOperations name="CreateNewMongooseModel"/>
    <eOperations name="validateModel"/>
    <eOperations name="exportModel"/>
    <eOperations name="exportValidation"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="modelName" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="confirmsTo" eType="#//dataset"
        containment="true"/>
  </eClassifiers>
</ecore:EPackage>
```

Appendix D: Creating the activity diagram in Papyrus

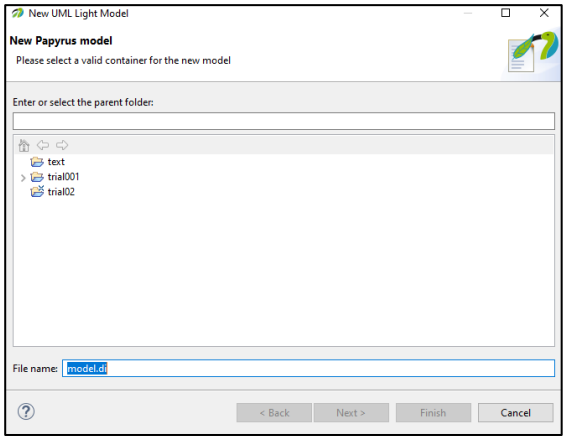
- Create a new UML project.

File → new → other … → Papyrus → UML Light Project



- Create a new UML model and select the generated project as its parent folder.

File → new → other … → Papyrus → UML Light Model (enter the project and model names)



- Initiate a new activity diagram.

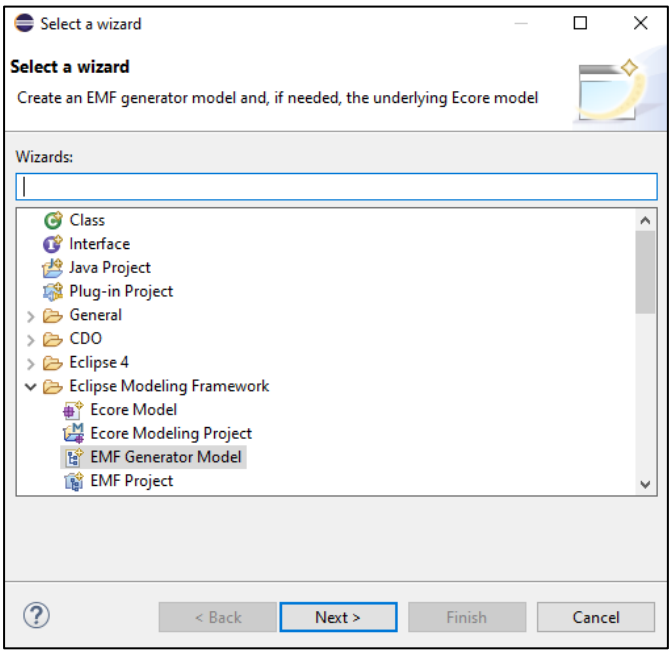
Open the generated model by double clicking on it → Papyrus → New Diagram → Light Activity Diagram

- Then use the elements in the palette at the right side of the application screen to create the required elements (nodes, flows, etc.)

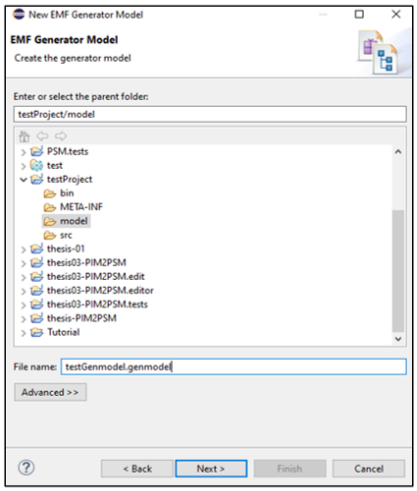
Appendix E: Generating the PIM from its metamodel

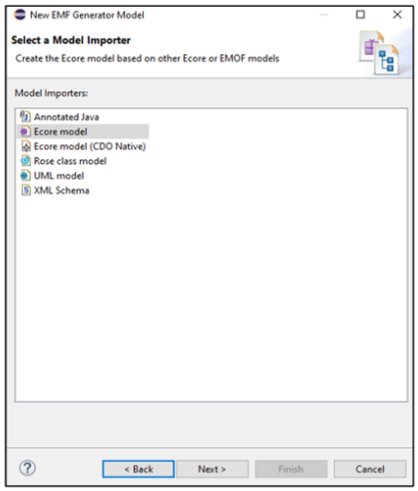
- First the model generator (.genmodel file) has to be created from the Ecore metamodel.

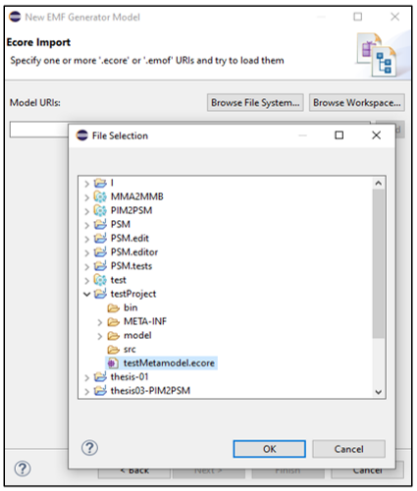
Right click on the model subfolder in the project folder → New → Other … → Eclipse Modeling Framework → EMF Generator Model



Select the project folder and enter a name for model generator. In the next window select "ecore model "option. In the next window from the "browse workspace …" button navigate to your project folder and select the Ecore metamodel you've developed before. Then Click finish in the next window.

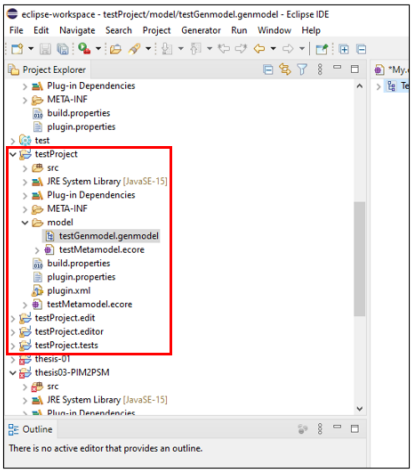






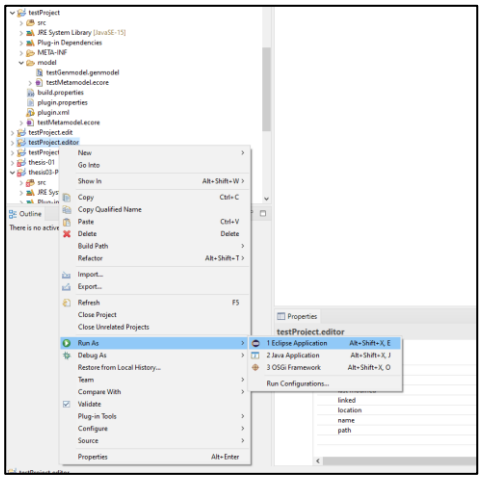
- The next step is generating the model, edit, and editor codes and plugins from the model generator.

Right click on the developed model generator → Generate All
*(If done properly, tree new folders with the names *.edit, *.editor, and *.test will be created automatically.)*



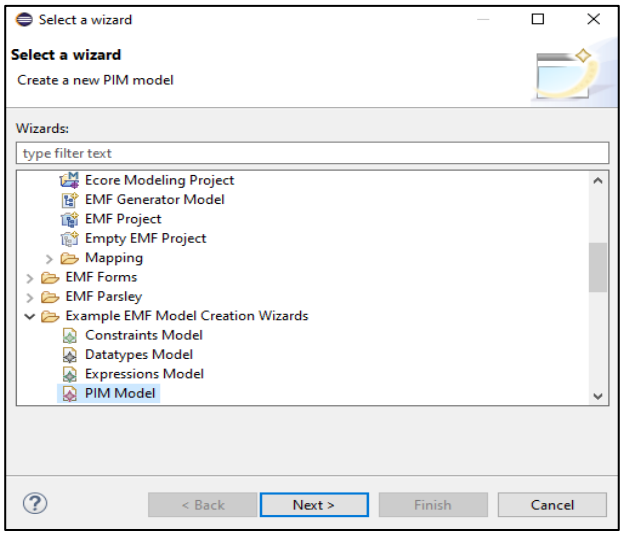
- Run the developed .editor plugin as Eclipse

Right click on .editor plugin folder → Run As Eclipse Application (a new Eclipse application will open)



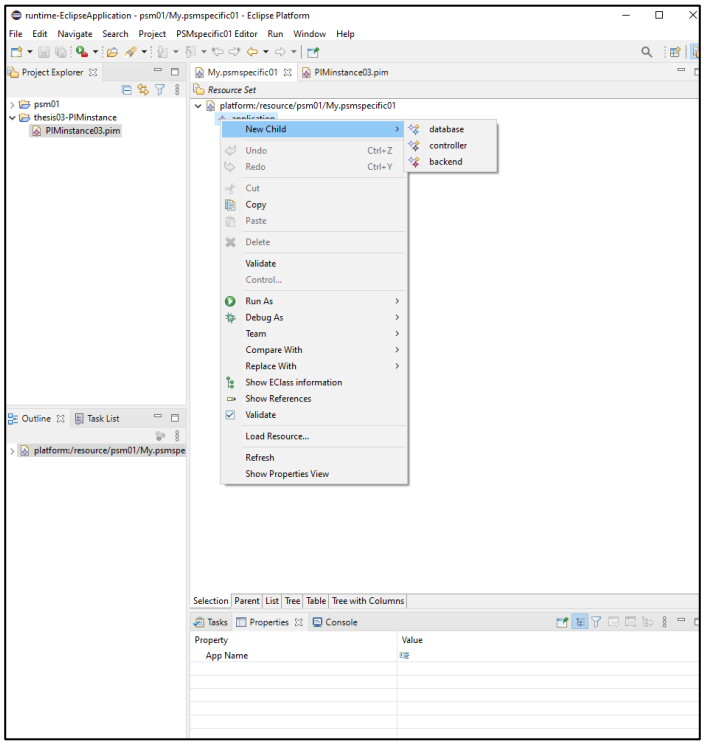
- Create the PIM in the newly opened Eclipse application window

First create an Eclipse modeling project
Then File → Other … → Example EMF Model Creation Wizards → Choose your previously developed metamodel



- Start adding elements, items, attributes, and properties to the model (considering the CIM)

Right click on the mode → New Childe → select item



Appendix F: The PIM in XML format

```
<?xml version="1.0" encoding="UTF-8"?>
<pim:APPLICATION xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:pim="http://pim/1.0.0" Title="PropertyFinder">
  <uses DBName="mongoDB">
    <has DCName="Collection1">
      <has DSName="usersData"/>
      <has DSName="propertiesData"/>
    </has>
  </uses>
  <includes>
    <has SName="usersBack">
      <worksWith DSName="usersData"/>
    </has>
    <has SName="propertiesBack">
      <worksWith DSName="propertiesData"/>
    </has>
  </includes>
  <has COName="frontEnd">
    <has/>
    <includes CName="mainTab">
      <isConnectedTo SName="propertiesBack">
        <worksWith DSName="propertiesData"/>
      </isConnectedTo>
      <has/>
      <hass/>
      <hasss/>
    </includes>
    <includes CName="propertiesTab">
      <isConnectedTo SName="propertiesBack"/>
    </includes>
    <includes CName="usersTab">
      <isConnectedTo SName="usersBack"/>
    </includes>
    <includes CName="loginTab">
      <isConnectedTo SName="usersBack"/>
    </includes>
  </has>
</pim:APPLICATION>
```