Journal of Theoretical and Applied Information Technology
30th June 2023. Vol.101. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ARE THERE POINTS OF SIMILARITY BETWEEN THE CLASSIC OBJECT-ORIENTED APPROACH AND THE MODEL PROGRAMMING APPROACH?

AZIZ SRAI[1], FATIMA GUEROUATE[2]

[1]ERCI2A, FSTH, Abdelmalek Essaadi University, Tetouan, Morocco

[2]LASTIMI Laboratory, Superior School of Technologies of Sale, Mohammadia School of engineering, Mohamed V University city of Rabat, Morocco

E-mail: [1]a.srai@uae.ac.ma, [2]guerouate@gmail.com

ABSTRACT

NoSQL databases (also known as Not Only SQL databases) are non-relational database systems used to store and retrieve data. Today, NoSQL databases are widely used in real-time web applications. NoSQL databases can also be referred to as big data databases or cloud databases. NoSQL databases are generally faster than SQL databases, so NoSQL databases are used for big data usage. In this article, we compare the two approaches, the classic approach (object-oriented development) and the new approach (model-driven approach or Model Driven Architecture), taking NoSQL databases as a case study. We first developed a software solution applied to the NoSQL database using the classic object-oriented approach using Spring Boot and secondly, we developed the same software solution through the use of the model approach or MDA in order to reconcile the two approaches in terms of implementing platform independence and in terms of development cycle time. We took as a case study, a use case of an eStore platform (class diagram). Considering an application in its entirety is a difficult task, which is why we first represented the platform with a simple class diagram.

Keywords: *Big Data, MDA approach, NoSQL, QVT, PIM model.*

1. INTRODUCTION

A significant number of NoSQL solutions have been developed. There are different ways to structure data, but the set of solutions developed can be divided into four models: the key-value model, the column-oriented model, the document-oriented model and the graph-oriented model. At their inception, these approaches were developed by relaxing relational principles. NoSQL is currently considered an add-on solution. In another context, the research world has recently developed an important approach called Model Driven Architecture (MDA). This approach, also called model programming, has shown a significant reduction in cost and time in terms of the application development cycle compared to other traditional development approaches (procedural approach, service-oriented approach, object-oriented approach, etc…). The contribution of this article is based on the demonstration, through a case study, of the advantages of the MDA approach compared to the classical object-oriented approach. We developed in parallel two solutions each with one of the approaches and we compared the two based on some criteria cited in the results section of this article.

2. LITERATURE REVIEW

Work [1] proposed an application of the MDA approach on E-learning platforms, the authors use UML to model the PIM, as well as ArcStyler for model transformation, then propose a UML profile for EJB. In this article, the transformation rules between the models have not been defined. The transformation rules between the two metamodels, source metamodel and target metamodel; remain a very important task to approve the validity of the MDA approach on a technology.

The author in [2] proposes a framework based on OMG's Model Driven Architecture. This platform-independent framework specifies and classify

existing and future Learning Management Systems (LMS).

The work of Bézivin et al. [3] is dedicated to the application of the MDA approach for the web services platform. In this work the authors presented a development of an illustrative example of e-business based on two different applications of a Model-Driven Architecture (MDA) approach. In the first application, the Platform Independent Model (PIM) is created using the Unified Modeling Language (UML). This PIM is transformed using Atlas Transformation Language (ATL) to generate the Platform Specific Model (PSM) based on three target platforms: Java, Web service and Java Web service developer pack (JWSDP). In the second application, the PIM is created using Enterprise Distributed Object Computing (EDOC) and transformed into another PSM based on the same target platforms.

The objective of work [4] is to generate a model respecting the n-tier architecture using the MDA approach; this model represents an E-learning application. The authors proposed two metamodels, a source metamodel based on UML, and a target metamodel based on an n-tier architecture. The authors have chosen the QVTo transformation language to perform all possible transformations between the two metamodels.

After a large state of the art that we have done, we have noticed that the majority of authors have focused their work on transformations between old technologies towards other more recent ones, while basing themselves on the MDA approach. We according to our knowledge, we did not find any work which presents a comparison based on the development in traditional approach and the MDA approach. In this work we are based on this track, i.e. which is more optimal in certain cases. Certainly, it is a difficult and complex task. However, as a first step in this context we have contributed this work to begin this path of research that links the two approaches. To do this, we developed a use case of a platform with both approaches, a classic development cycle, and programming by model, and we compared the two. We have gathered the characteristics of each in this article.

3. METHODOLOGY

Software engineering is currently moving towards model engineering, after the object approach, where each software artifact is considered as a model. Engineering led by MDE models (Model Driven Engineering) presents a development approach that has become very popular in software engineering, which focuses on the creation and exploitation of abstract models. In other words, it is an abstract representation of the knowledge and activities that govern a particular application domain facilitating the understanding of the modeled system. It describes all the concepts and technologies around model management. The specification phase is particularly important in an MDE approach and represents a substantial part of the development cycle. This allows developers to focus on the desired behavior of the system, without worrying about how to implement it. The implementation phase is then started at the end of the cycle, once the specification is completed and validated. The partial generation of low-level code from the specification also reduces development time and therefore development costs. The MDE approach aims to generate all or part of the application from models. This in itself increases productivity while optimizing compatibility between systems through the large-scale reuse of standardized models. This simplifies the design process, and promotes communication between individuals and teams working on a system through standardization of terminologies and best practices used in the field of software engineering.

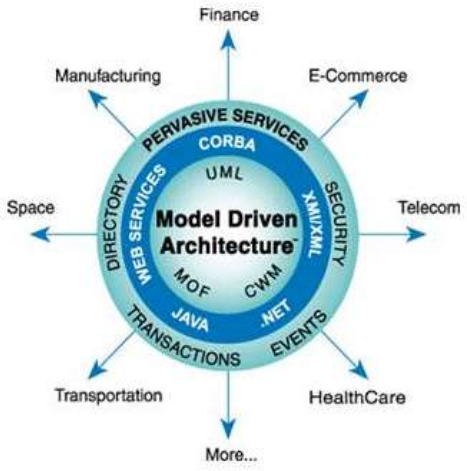


Figure 1: Diagram summarizing the languages and methods provided by MDA architecture

The MDE approach aims to generate all or part of the application from models. This in itself increases productivity while optimizing compatibility between systems through the large-scale reuse of standardized models. This simplifies the design process, and promotes communication between individuals and teams working on a system through

standardization of terminologies and best practices used in the field of software engineering.

3.1 Principles of model-driven engineering

In recent years, many organizations are interested in model-oriented engineering because it is an approach that provides the basis for using models to better understand, design, build, deploy, maintain and modify a system. This is a beneficial development for several reasons. First, the MDE approach encourages the effective use of models in the software development process, and second, it supports the reuse of best practices when creating a system. Its main goals are portability, interoperability and reusability through the separation of platform-dependent aspects from more abstract application-independent aspects. This architecture was introduced and defined by the OMG and is thus in line with the continuity of the object-oriented approach, by increasing the level of abstraction to a level where another set of concepts and relations is used, saying model. The model is an essential condition for the realization of a solid architecture as well as a good understanding of the system to be developed. It is an often partial specification of the functionality, structure and behavior of a system. This model can be considered as an abstraction of a system modeled as a set of facts, subsequently expressed in a clearly defined modeling language to formulate what is called a metamodel. Several MDE approaches have emerged. The most popular of these approaches is the OMG Model Driven Architecture. This has been extended and is not limited to models, but rather puts the models and not the programs. This is called Model Oriented Architecture (MDA).

3.2 Model-Driven Architecture

Model Driven Architecture (MDA) is a model-oriented approach defined by the Object Management Group (OMG) and made public at the end of 2000. This approach specifies three levels of abstraction (level business, platform independent level and platform dependent level) for the description of a system architecture and proposes a development process based on these three levels and driven by models. The basic idea of MDA is to separate the functional specifications of a system from the specifications of its implementation on a given target platform. In an MDA oriented development process, everything is seen as a model. Thus, MDA identifies four types of models: CIM, PIM, PDM and PSM. The CIM (Computation Independent Model), also called domain model or business model, models the requirements of the system. Its purpose is to help understand the problem but also to fix a common vocabulary for a particular field. MDA makes no recommendation as to the language to be used to describe CIMs. The PIM (Platform Independent Model) or analysis and abstract design model of the application. This model describes the system without showing the details of its use on a particular platform. In MDA, it is possible to build several PIM models independent of the target platform. PIM models do not contain any information about the execution platforms. A PIM must be refined with the details of one or more particular architectures to produce a PSM. The PDM (Platform Description Model) describes the platform on which the system will be run (for example component models at different levels of abstraction like CCM or EJB).The PSM (Platform Specific Model) or specific model of execution platforms.PSM is the model produced by transforming a PIM to take into account technical information relating to the chosen platform. PSM can be refined by successive transformations until an executable system is obtained. The figure below gives an overview of an MDA process commonly called the Y development cycle by showing the different levels of abstraction associated with models.

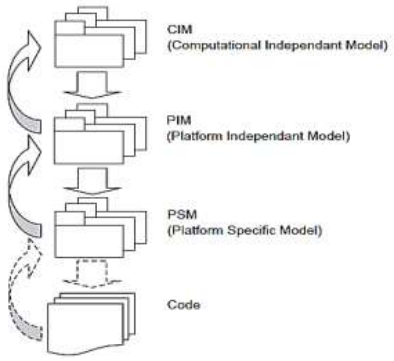


Figure 2: Principles of the MDA process

3.3 Model transformation

MDA architecture is a means of structuring and managing the software architecture of an organization where models are used on a large scale. Indeed, it allows defining a structuring of guidelines for specifications expressed as models. It is supported by automated tools and services to both define the models and facilitate transformations between different model types. In other words, MDA makes it possible to separate the

specification of the functionalities of the studied system from the specification of its application on its implementation platform. It offers the possibility of designing models independent of all platforms or implementation environments. The MDA approach provides a means through models to guide the understanding, design, the deployment, operation and maintenance of the studied system.

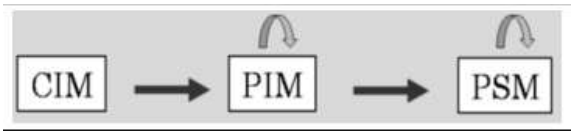


Figure 3: structure of a model transformation

In the cycle of the implementation of the MDA architecture, three groups of models have been specified, CIM (Computation-Independent Model) processing-independent models: the most abstract model in the MDA approach. It represents the context and purpose of the model without any complexity of control. Platform-independent models PIM (Platform-Independent Model): the model describes the behavior and structure of the application independently of the platform implemented. Platform-specific models PSM (Platform-Specific Model): the model cannot be executed but it contains all the necessary information, concerning a specific platform, those developers can use for the implementation.

3.4 Unified Modeling Language (UML)

UML is a language standardized by the OMG allowing to model a system according to different points of view, static and dynamic. The static structure makes it possible to model a system using objects, attributes, operations and relations. The dynamic structure makes it possible to model the dynamic behavior of a system by showing the interactions between objects or the changes of state within an object. State and activity diagrams fall into this category.

3.5 History of software development approaches

The evolution of software technology continues to be dynamic. New tools and techniques are continually being announced in rapid succession. This has forced the computer industry and players in this sector to constantly seek new approaches to software design and development.

3.5.1 The procedural approach

Since the invention of computers, many development approaches have been tried and tested. With the advent of programming languages like C, structured programming became very popular this was the main development technique in the 1980s. It gives programmers the power to write moderately complex program quite easily. The main features presented in The Procedural Approach are: Most functions share global data. Thus, data moves freely from one function to another and transforms it from one form to another. The program design uses a top-down approach. Big problems are broken down into smaller ones and mapped to functions. The maintenance of systems developed with the procedural approach is difficult and expensive. A more modular development approach became necessary. This is how the object-oriented approach was born.

3.5.2 The object approach

The limitations presented by the procedural approach led to the appearance of the object approach. The main objective of this approach is to provide a better programming model for representing the world. The basis of the object approach is the concept of object. An object is defined as a software entity inspired by the real world. An object has specific properties and provides access methods. An object is an instance of a class. Moreover, an object supports the concepts of abstraction, encapsulation, inheritance and polymorphism. These properties are the main concepts of the object approach.

3.5.3 The component approach

The need for component-based development has arisen to increase the level of abstraction and to change the way of developing computer systems. The component-oriented paradigm has enabled several improvements over the object-oriented paradigm, including increased productivity, improved quality, and reduced development. The component approach is based on the idea of developing applications using components. Components are analogous to functions in procedure-oriented development. A component is a more abstract form and it is capable of performing specific functionality. Moreover, a component is a software object intended to interact with other components, encapsulating certain functionality or a set of functionalities. A component must have a

clearly defined interface through which it is associated with other components and conforms to a prescribed behavior common to all components of an architecture. Component-based software systems are developed by selecting appropriate standard components and then assembling them with well-defined software architecture. The component approach aims to create a software package in such a way that its components can be easily reused in other similar or different applications. The use of middleware aims to assemble the components and make them interoperable.

Middleware such as CORBA (Common Object Request Broker Architecture), Java RMI (Java Remote Method Invocation) and DCOM (Distributed Component Object Model) have been proposed to simplify the development of software based on a component approach, and to allow interoperability between distributed and heterogeneous systems. They responded quite well to interoperability in the context of distributed systems, but the advent of the Internet introduced new requirements to which they were not suited. These middleware have been extended to meet these new requirements, but with little success.

3.5.4 The service approach

Both the service approach and the component approach seem to have the same objective: to provide a basis for highly integrated and highly interoperable software architecture, allowing efficient and error-free software development. So, develop a type of architecture allowing weak coupling and high reusability of its components. There is no clear demarcation between the service approach and the component approach. In principle, the service approach is the improvement of the component approach because Individual services are unique components, which can be linked to achieve new business logic, new services, or a new component. In principle, a service is defined as a software entity whose use must be made within a Service-oriented Architecture (SOA). The idea of the service-oriented approach appeared in the 2000s. The historical sequence of the four approaches that we have just described succinctly can be found in the extended Racoon wave diagram shown in Figure 4.

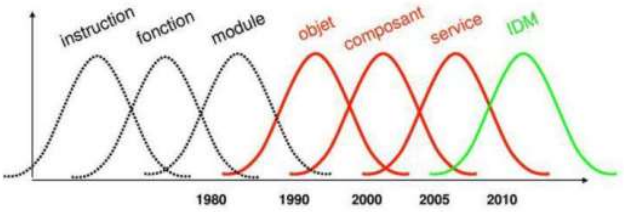


Figure 4: Extended Racoon wave diagram [Donsez, 2006].

3.6 QVT language

Query/View/Transformation (QVT) is a standard defined by the OMG to specify transformations between models, whose meta model satisfies the MOF standard. It includes a declarative part and an imperative part. The declarative part consists of two parts: a part carrying out the correspondence between the two models expressed in the standard MOF named QVTr (relations), and a part which makes it possible to evaluate conditions on the elements of our models to make them correspond, named QVTc (Core). These two parts use OCL (Object Constraint Language) to define the matching rules. OCL is a formal language standardized by the OMG for specifying software constraints. The imperative part, consisting of QVTo (operational), makes it possible to extend the declarative language. Constructs such as for loops or if conditions are offered there. QVTo also introduces the use of imperative OCL rules.

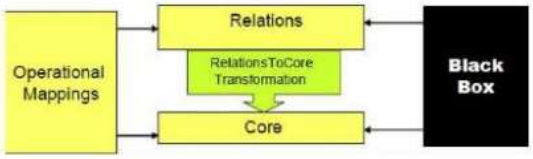


Figure 5: Relationships between QVT metamodels

3.7 Graph-oriented model

The graph-oriented model is based on graph theory. The graph-oriented model is based on three concepts; node, relation and property. Each node has properties. Relationships connect nodes and optionally have properties. This type of approach facilitates navigational queries between nodes by following the relationships between them: each node has a physical pointer to neighboring nodes allowing fast local search.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

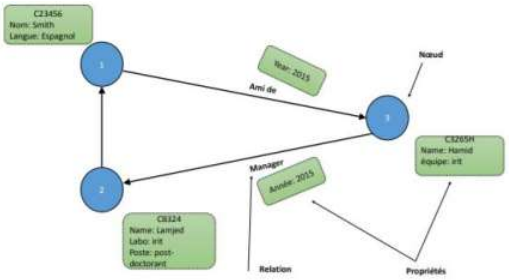


Figure 6: Principle of the graph-oriented model

The structure of the graph-oriented model is very suitable for responding to issues such as the management of a company's social network or any other storage requiring graph browsing. The graph-oriented model is also characterized by schema flexibility; it is not any need to create a schematic first for nodes and relations.

3.8 Source and Target Metamodels

The work presented in this article is a work aiming at the comparison in certain criteria between the classic object-oriented approach and the new MDA approach, also known as model-based programming. We have developed an application with both approaches in parallel and we have compared the two in terms of development time and certain technical difficulties. For the MDA approach we have developed two metamodels, a source metamodel and a target metamodel.

3.9 UML source Metamodel

Figure 7, illustrates the simplified UML source meta-model based on packages including operations, associations and classes. Those classes are composed of properties with parameters.

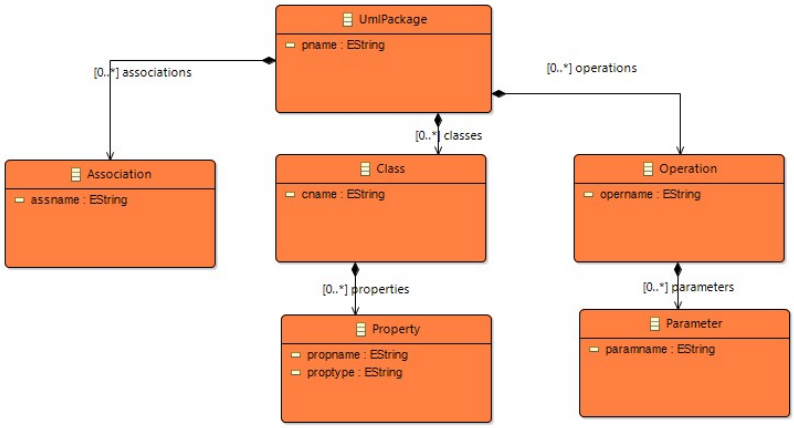


Figure 7: UML source meta-model

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.10 Document oriented target Metamodel

Figure 8, illustrates the simplified Document target meta-model:

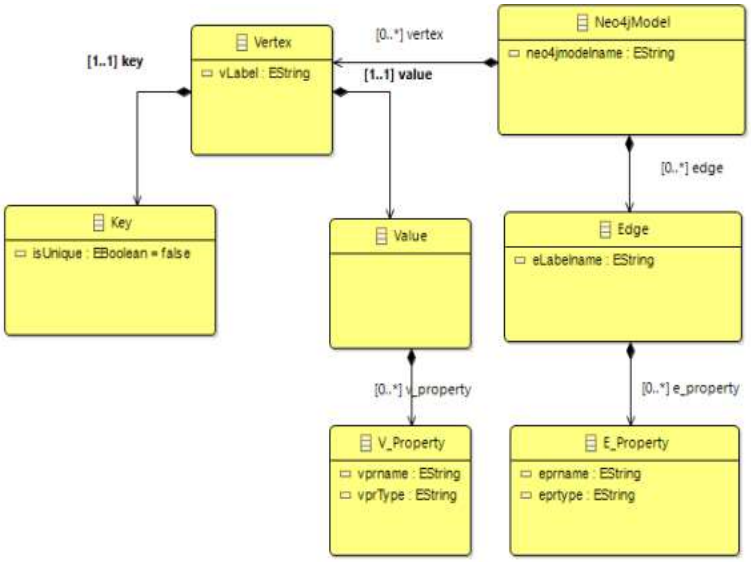


Figure 8: Graph target meta-model

4. RESULTS AND DISCUSSIONS

In this section, the results of research are explained and at the same time is given the comprehensive discussion. Results can be presented in figures, graphs, tables and others that make the reader understand easily. The discussion can be made in several sub-chapters. It is strongly suggested that comparison wih results from other published articles are provided to give more context and to strengthen the claim of novelty.

In this part, we will show the different phases of development of our case study with the classical object oriented approach, we recall that our case study is a class diagram made up of two class society and person with working associations, we suppose that this case study is sufficient to apply it in our study on this article. We have developed this case study respecting all the development cycle from design to unit testing. We have considered the source model (Figure, 9) that we assume is sufficient to mount our work.

4.1 Development of the case study with the classical object oriented approach

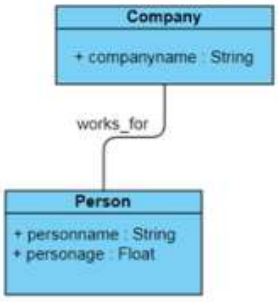


Figure 9: Class diagram representing the case study

```java
package com.springboot.springbootMDAmultilayredapplication.model;
@NodeEntity
@Data
@AllArgsConstructor
@NoArgsConstructor
@EqualsAndHashCode(exclude="entityId")
public class Person {
    @Id
    @GeneratedValue
    private Long entityId;
    private String name;

}
```

```java
package com.springboot.springbootMDAmultilayredapplication.model;

@NodeEntity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Company {
    @Id
    @GeneratedValue
    private Long entityId;
    private String name;
    private String location;

    @Relationship(type="works_for",direction=Relationship.INCOMING)
    private Set<Person> persons;
}
```

```java
@RestController
@RequestMapping(value="/api/company")
public class CompanyController {
    private CompanyService service;
@PostMapping
private Company saveCompany(@RequestBody Company company){
    return service.saveCompany(company);
}
}
```

```java
@Service
public class CompanyService {
    @Autowired
    private CompanyRepository companyRepository;
    @Autowired
    PersonRepository personRepository;
    public Company saveCompany(Company company){
        return companyRepository.save(company);
    }

}
```

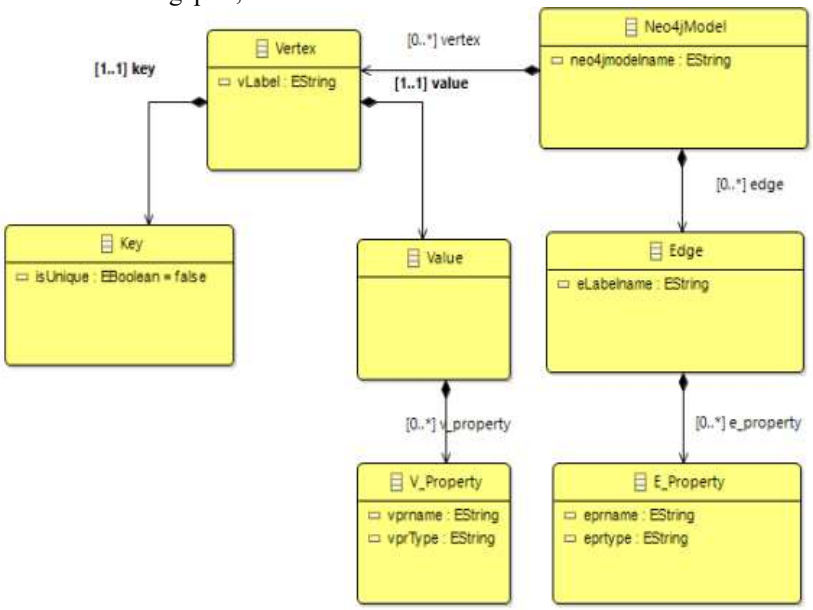
```java
package com.springboot.springbootMDAmultilayredapplication.config;

import org.neo4j.driver.internal.SessionFactory;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.neo4j.repository.config.EnableNeo4jRepositories;
import org.springframework.transaction.annotation.EnableTransactionManagement;
import org.springframework.web.servlet.support.SessionFlashMapManager;

@Configuration
@EnableNeo4jRepositories(basePackageClasses="com.springboot.springbootMDAmultilayredapplication.repository")
@EnableTransactionManagement
public class Neo4jConfig {
    @Bean
    public SessionFactory (){
        return new SessionFactory(Configuration(),"com.springboot.springbootMDAmultilayredapplication.model");
    }
    @Bean
    public org.neo4j.ogm.config.Configuration configuration(){
        org.neo4j.ogm.config.Configuration configuration =new org.neo4j.ogm.config.Configuration.Builder()
                .uri("http://localhost:7687")
                .credentials("neo4j","password")
                .autoIndex("none")
                .build();
        return configuration;
    }
    @Bean
    public Neao4jTransactionManager transactionManager(){
        return new Neo4jTransactionManager(sessionFactory());
    }
```

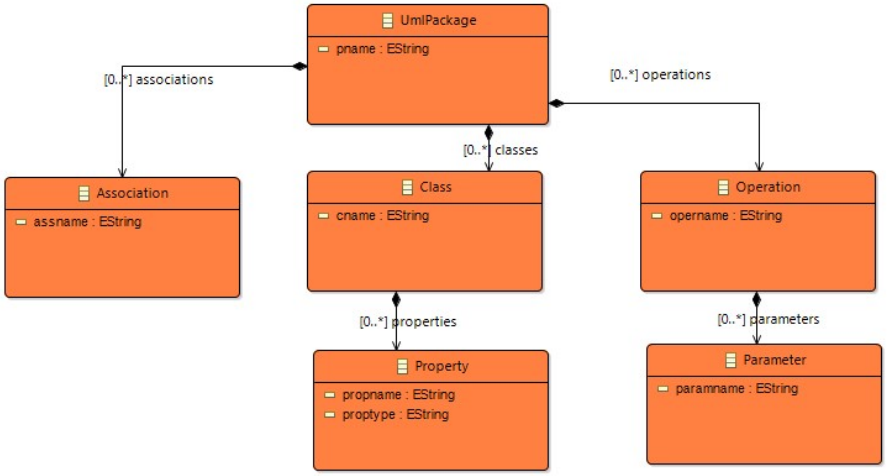
Figure 10: Pieces of code that demonstrate the classic development of our case study

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4.2 Development of the case study with the model approach (MDA approach)

In this approach, we will develop our case study with the MDA or model programming approach through successive transformations between models, from the PIM model to the PSM model. We compare in the following part, the difference between this approach and the classic approach. We also demonstrate through this scientific contribution the adoption of this approach compared to other approaches. We detail below the different studies of the development of our work through the MDA approach.





ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

```
modeltype umlGraph uses umlGraph("http://umlGraph.mm");
modeltype neo4j uses neo4j("http://neo4j.mm");

transformation transf1(in source:umlGraph, out target:neo4j);

main() {

      source.rootObjects()[UmlPackage]->map UmlPackageToneo4jmodelname();
}



mapping UmlPackage :: UmlPackageToneo4jmodelname() : Neo4jModel
      {
     result.neo4jmodelname := self.pname;
     vertex:=source.rootObjects()[Class]->map ClassToVertex();
     result.vertex.value.v_property=source.rootObjects()[Property]->map ClassPropertyToVertex_Property();

}
mapping Class :: ClassToVertex() : Vertex
      {
     result.vLabel := self.cname;
     result.value.v_property+=self.properties-> map ClassPropertyToVertex_Property();

}
mapping Property :: ClassPropertyToVertex_Property() : V_Property
      {
     result.vprname := self.propname;
     result.vprType := self.proptype;
}
```

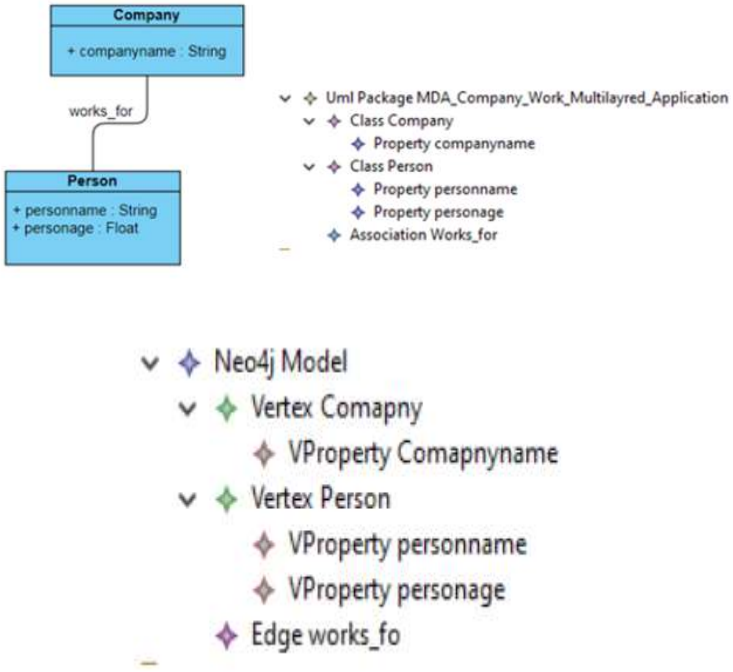


Figure 11: Different phases of the development of our case study through the MDA approach

4.3 Comparison between the two approaches

In this work we have developed a case study with two approaches, a classical object oriented approach and an MDA approach. During all the development cycle of the two approaches we discovered that the adoption of the MDA approach compared to the classic development approaches is too recommended for several reasons, among these major reasons we can retain a complete independence of the MDA approach in relation to

the implementation platform of the solution thing that is necessary in the classical development where the choice of the implementation platform is essential. Another major factor is the development time, we discovered that the MDA approach requires an average time during the development cycle, on the other hand the classic development requires a remarkable time during the development cycle. We reason in the table below these factors.

TABLE 1: A TABLE WHICH SUMMARIZES the comparison between the two approaches.

Features	Independence from the implementation platform	Time in relation to the development cycle
classic approach	Yes This approach is 100% linked to the implementation platform	A heavy development cycle
Model Driven Architecture approach	No This approach is not linked to the implementation platform, but in some cases it can be linked with certain percentage, because the MDA approach presents to generate part or all of the solution	A light development cycle

5. CONCLUSION

In this work, we presented a development of a case study for an eStore application (simple class diagram), with two approaches, a classic object-oriented approach and a search approach, which is the MDA approach. We have compared the two approaches through this scientific contribution. We have demonstrated the validity of the MDA approach and the advantages it has over the object-oriented approach in the development of software solutions. As perspectives of this work, we try to apply the MDA approach on other platforms such as NoSQL databases, mobile, web and desktop applications. We support model programming through these research studies.

REFERENCES:

[1] Hong Wang, Dong Zhang, Jun Zhou, MDA-based development of e-learning system, IEEE Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003, DOI: 10.1109/CMPSAC.2003.1245417.

[2] Kurillova,Model Driven E-Learning Platform Integration, Proceedings of the EC-TEL 2007 PROLEARN Doctoral Consortium, Crete, Greece, September 18, 2007.

[3] J. Bezivin; S. Hammoudi; D. Lopes; F. Jouault, Applying MDA approach for Web service platform, Proceedings. Eighth IEEE International Enterprise Distributed Object Computing Conference, 2004. EDOC 2004, Monterey, CA, USA, DOI: 10.1109/EDOC.2004.1342505.

[4] A. Srai, F. Guerouate, N. Berbiche, H. Drissi, "Generated PSM Web Model for E-learning Platform Respecting n-tiers Architecture," International Journal of Emerging Technologies in Learning (iJET), vol. 12, no. 10, pp. 212-220, 2017.

[5] Toward Automatic Generation of Column-Oriented NoSQL Databases in Big Data Context, Redouane Esbai, Fouad Elotmani, Fatima Zahra Belkadi, International Journal of Online and Biomedical Engineering, iJOE – Vol. 15, No. 9, 2019.

[6] Liliana Favre, Liliana Martinez, Claudia Pereira, "Modernizing software in science and engineering: From C/C++ applications to mobile platforms," Conference: ECCOMAS Congress 2016 European Congress on Computational Methods in Applied Sciences and Engineering, DOI: 10.7712/100016.2402.4906.

[7] Kim, S.D., Min, H.G., Her, J.S., Chang, S.H.: Dream : a practical product line engineering using model driven architecture. In: Proceedings of the Third International

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Conference on Information Technology and Applications (ICITA 2005) (2005).

[8] Czarnecki, K., Helsen, S., Classification of Model Transformation Approaches, in online proceedings of the 2nd OOPSLA'03 Workshop on Generative Techniques in the Context of MDA. Anaheim, October, 2003.

[9] Donsez, D., "Objets, composants et services : intégration de propriétés non fonctionnelles", Habilitation à Diriger des Recherches de l'Université Joseph Fourier, http://wwwadele.imag.fr/users/Didier.Donsez/pub/publi/hdr/, Déc. 2006.