computers

MDPI

Article

MoDAr-WA: Tool Support to Automate an MDA Approach for MVC Web Application

Imane Essebaa *, Salima Chantit and Mohammed Ramdani

Computer Science Laboratory of Mohammedia, Faculty of Sciences and Technologies of Mohammedia, Hassan 2 University of Casablanca, Casablanca 20000, Morocco; salima.chantit@gmail.com (S.C.); ramdani@fstm.ac.ma (M.R.)

* Correspondce: imane.essebaa@gmail.com

Received: 31 October 2019; Accepted: 1 December 2019; Published: 5 December 2019

Abstract: Model-driven engineering (MDE) uses models during the application development process. Thus, the MDE is particularly based on model-driven architecture (MDA), which is one of the important variants of the Object Management Group (OMG). MDA aims to generate source code from abstract models through several model transformations between, and inside the different MDA levels: computation independent model (CIM), platform independent model (PIM), and platform specific model (PSM) before code. In this context, several methods and tools were proposed in the literature and in the industry that aim to automatically generate the source code from the MDA levels. However, researchers still meet many constraints—model specifications, transformation automation, and level traceability. In this paper, we present a tool support, the model-driven architecture for web application (MoDAr-WA), that implements our proposed approach, aiming to automate transformations from the highest MDA level (CIM) to the lowest one (code) to ensure traceability. This paper is a continuity of our previous works, where we automate transformation from the CIM level to the PIM level. For this aim, we present a set of meta-models, QVT and Acceleo transformations, as well as the tools used to develop our Eclipse plug-in, MoDAr-WA. In particular, we used QVT rules for transformations between models and Acceleo for generating code from models. Finally, we use MoDAr-WA to apply the proposed approach to the MusicStore system case study and compare the generated code from CIM to the original application code.

Keywords: model-driven architecture; platform independent model; platform specific model; model-to-model transformations; model-to-text transformations; Eclipse plug-in; generated model-view-controller source code

1. Introduction

The continuous advance on software engineering technologies contributed to the improvement of model-driven software engineering (MDSE) approaches to enhance the software engineering lifecycle [1]. Indeed, the emergence of the model-driven architecture (MDA) standard, defined in 2001 by the Object Management Group (OMG), has contributed to this improvement.

Three levels of abstraction are defined in MDA that describe the steps of software development in the context of model-driven engineering (MDE). System features are defined in the computation independent model (CIM) level and described using an appropriate DSL (Domain Specific Language), for example UML(Unified Modeling Language), BPMN(Business Process Models and Notations), or even natural language or a structural one like the semantic business vocabulary and business rules (SBVR) standard. The CIM level is then transformed into the platform independent model (PIM) level, which is in its turn transformed into the platform specific model (PSM) level by means of successive transformations to finally generate the source code according to different implementation platforms (e.g., Java, .NET, etc.). CIMs, PIMs, and PSMs are defined using modeling languages that

are described using a corresponding meta-model and transformations are (can be) described using transformation meta-model.

A software model is of a higher abstraction level than code, thus, it constitutes a bridge among project stakeholders to reason and communicate about the software system. However, in the major cases, coding and testing are the main activities in software engineering, indeed, models are typically discarded after finishing their role of reasoning and communicating the requirements.

The increasing demand for rigorous development practices in many domains, including web applications, requires an evolution of software system principles, including model-driven engineering ones [2]. Despite the number of model-driven approaches for web applications development that have been developed over the last decade (e.g., UWE [3], WebML [4], AndroMDA [5], just to mention a few), the traceability and automation of these approaches are still an issue. Besides, none of them consider the entire MDA process from CIM to code.

The objective of this paper is to propose an approach and present a tool, the support model-driven architecture for web application (MoDAr-WA) to automate the generation of MVC web application code from the CIM model description. In particular, our proposed approach in this work aims to address the following research questions:

- What aspects of each MDA level should the system designer cover while modeling the software system?
- Which models should be chosen to model the system well at each MDA level?
- How can we generate the code architecture from the CIM level using successive transformations between the different MDA levels with respect to traceability?

In this paper, we propose an approach to automate transformations between PIM and PSM, and then the code generation in the context of MDA. This approach is a continuity of our previous works where we automate transformations from the CIM to the PIM level [6]. In this approach, we define, according to OMG, how each level should be modeled, and present transformation rules that were implemented as an Eclipse plug-in, (MoDAr-WA) to ensure the transition between MDA levels and the code generation.

To achieve this operation of code generation, we used QVT language to transform models from CIM to PIM then to PSM and Acceleo to generate the code application that respect the MVC architecture from PSM models.

This paper is organized as follows: Section 2 presents our proposed approach as one of the solutions to deal with the described problems, and its implementation is detailed in Section 3. The illustration of our approach on a case study of an E-commerce application is given in Section 4. Then, in Section 5, we discuss and compare existing approaches that were conducted in the [6] same context as our proposed approach, and finally, we conclude in the last section with a presentation of our future works.

2. Proposed Approach

In this section, we present our approach to transforming PIM models to PSM ones that respect MVC architecture, then the transformations that generate the application source code. This approach is a continuity of our previous works [6,7] that were dedicated to automating transformations between CIM and PIM levels of MDA. This approach consists of:

- Describing system requirements using business vocabulary and business rules of SBVR in CIM level [7].
- Generating automatically the use case diagram from SBVR in the same level [7] using the MoDAr-WA plug-in.
- Generating the PIM level automatically, which is represented by the business class diagram and system sequence diagram from the CIM level using the QVT transformation rules implemented in the plug-in [6].

- Automatically generating the PSM level of the MVC web application modeled by detailed class diagram and detailed sequence diagram using the MoDAr-WA plug-in (presented in this paper).
- Automatically generating the application code through the Acceleo transformation rules implemented in the plug-in (presented in this paper).

The Figure 1 below describes an overview of our approach.

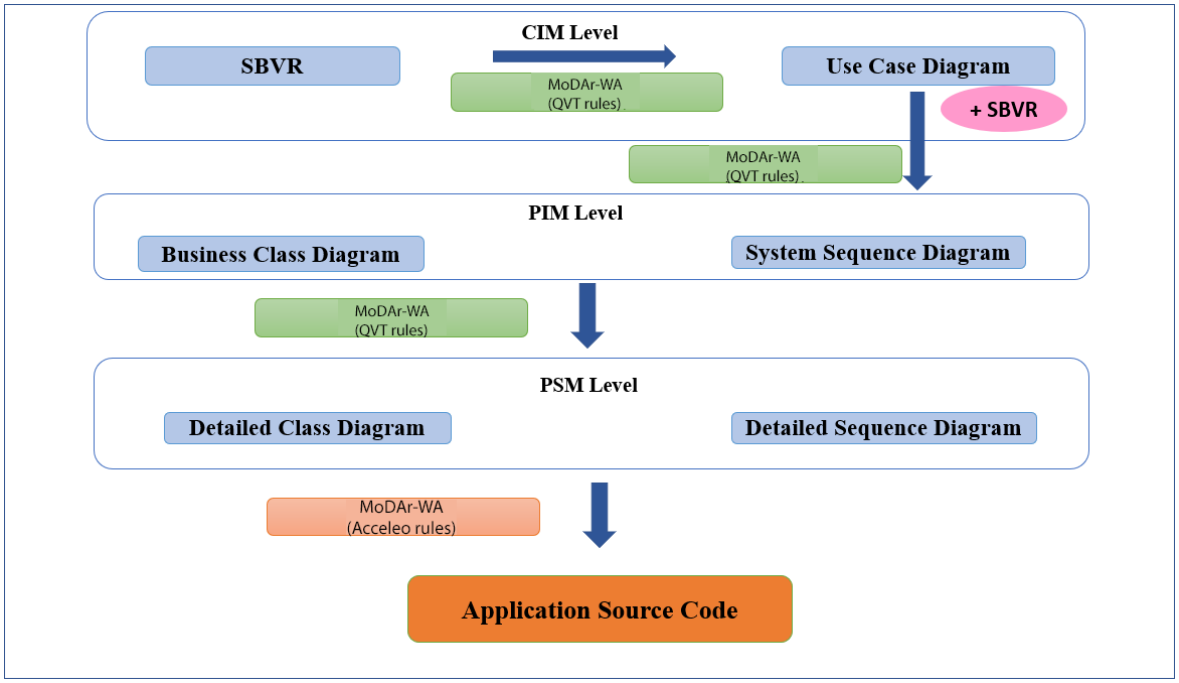

Figure 1. Overview of our model-driven architecture (MDA) transformational approach.

As mentioned before, this paper is a continuity of papers [6,7] that provide transformations between CIM and PIM levels.

The following section focuses on the main elements (meta-models and transformations) of generating the MVC application code from the IM level going through the PSM level.

2.1. PIM Level in Our Approach

The platform independent model, also called the analysis and design model, describes the "What" of the system independently of any technical information about the execution platform. It defines the business logic of the system regardless of their technical details. The main aspects that must be covered to model the PIM level well are the structural and behavioral aspects.

- The structural aspect describes the link between elements representing the system. This aspect is covered by the business class diagram in our approach. Figure 2 represents the main fragments of the business class diagram.
- The behavioral aspect describes the flow of actions between system elements. In our approach, this aspect is covered by the system sequence diagram which is a type of sequence diagram where the system is targeted as a black-box in order to respect the platform independence criteria of the PIM level. Figure 3 represents the main fragments of system sequence diagram.

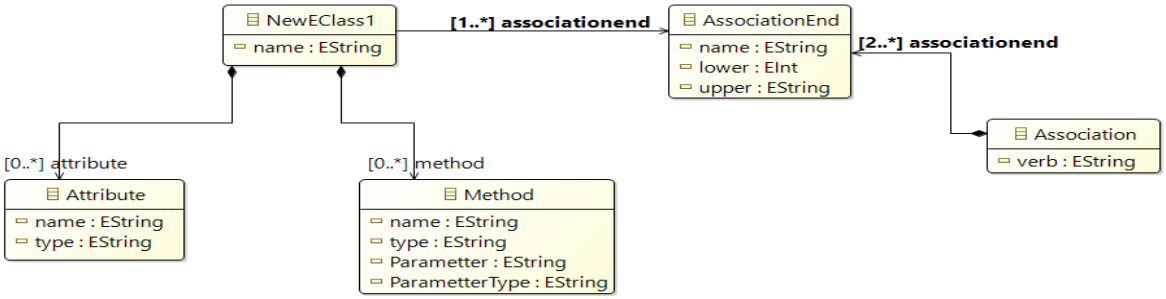

Figure 2. Main elements of the business class diagram in our approach.

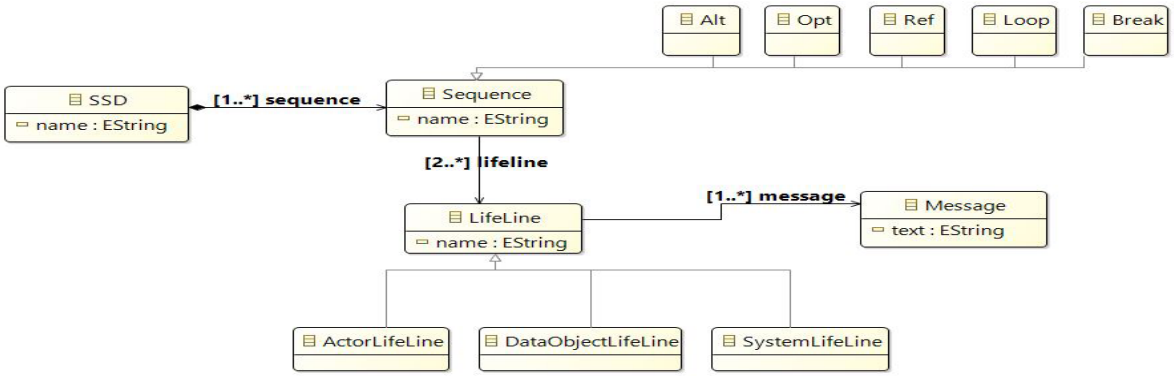

Figure 3. Main elements of the system sequence diagram in our approach.

2.2. PSM in Our Approach

The Platform specific model, also called the code model, is the level of concrete design and the last level in MDA before code. PSM is the result the combination of both the PIM and technical details that will be transformed into code. The defining code model is considered the most delicate step in MDA, as it requires the consideration of different aspects to generate code for a specific platform. According to OMG specifications, four aspects can be defined for the PSM level:

- Static Aspect: Describes static elements of the PSM level. In our approach this aspect is represented by the model classes in the detailed class diagram.
- Structural Aspect: Describes associations between class diagram elements. In our approach, this aspect is covered by associations in the detailed class diagram.
- Dynamic Aspect: Describes the communication between different elements of the system. Controller classes cover this aspect in our approach.
- Behavioral Aspect: Describes the flow of actions in the system. In our approach, this aspect is represented by the detailed sequence diagram, which is a detailed version of the system sequence diagram, where the system is replaced by all internal system objects and messages inside the system. In our PSM level, we model each layer of MVC architecture as lifeline elements in DSD. Figure 4 describes the main fragments of the detailed sequence diagram.

Figure 5 describes the main fragments of detailed class diagram (DCD) which is used in our approach to cover the different MVC layers that are represented as a "Role" of the class element.

2.3. Transformations Rules

In this section, we will define the different transformation rules that allow the automatic transformation from the CIM level to code. The proposed transformation process takes, as input, the source models, applies the transformation rules and then generates target models. This process is conformed to the model object facility (MOF), which is an OMG standard that allows the describing

of meta-models and their manipulation. This section is organized into two parts, PIM to PSM transformation rules and PSM to source code transformation rules.

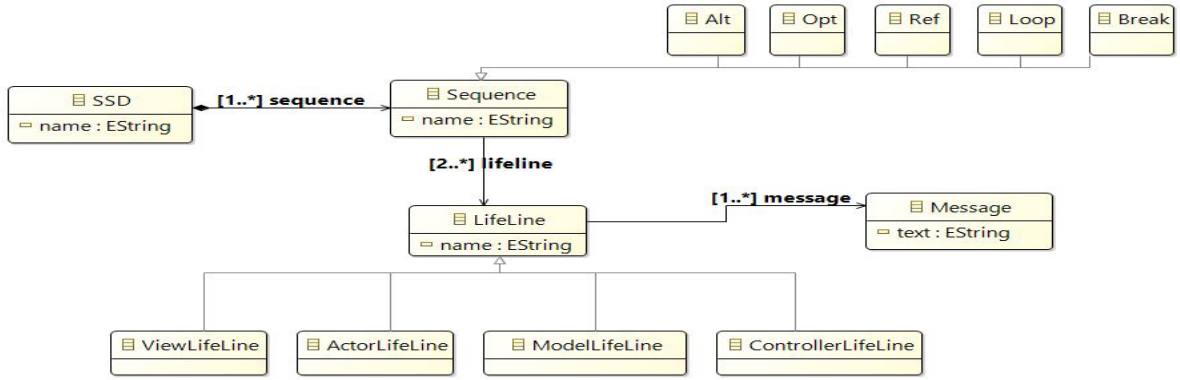

Figure 4. Main elements of the detailed sequence diagram in our approach.

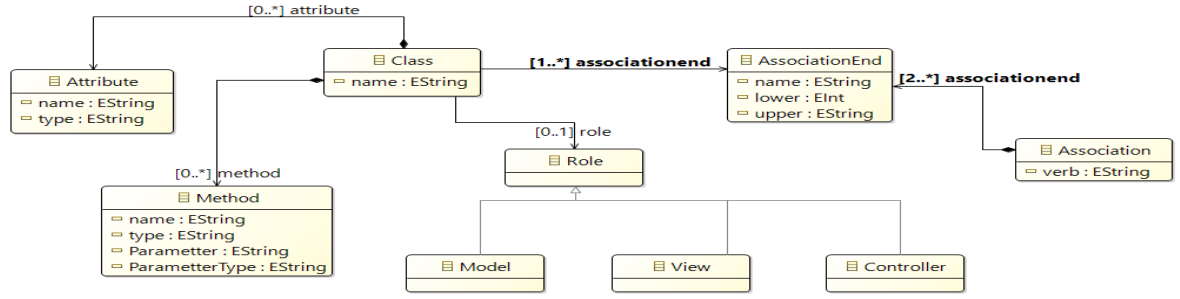

Figure 5. Main elements of the detailed class diagram in our approach.

2.3.1. PIM to PSM Transformation Rules

To automatically generate the PSM level from the PIM level, we apply two types of model-to-model transformations:

- **BCD&SSD2MVC_DCD**: The business class diagram and system sequence diagram to MVC class diagram is a set of transformation rules that generate the detailed class diagram of the PSM level from BCD (Business Class Diagram) and SSD (System Sequence Diagram) of the PIM level.

This set of rules contains sixteen rules that are described in Table 1.

1. Class2Model: Each class in BCD in the PIM level (generated from Actor and DataObject of the CIM) is turned into a model class in MVC detailed class diagram in the PSM level.
2. Class.name2Model.name: The name of the model is the same name as the class from which the model was generated, preceded by the stereotype «Model».
3. Class.Attribute2Model.Attribute: Attribute elements of the model are the same as the attributes of the class in the PIM level with same names and types (these attributes were generated from "Is_property_of" fact type in the CIM level).
4. Class.Method2Model.Method: The class' methods are transformed into the methods of the model. At this level, the model class contains only the methods' signatures that call the methods described in the controller class.
5. Association2Controller: The association between two classes in the PIM level will be transformed into a controller class in the PSM level (the associations in the PIM level were generated from the "Associative" fact type in the CIM level).
6. Association.verb2Controller.name: The name of the controller class is generated from the verb of the association preceded by <Controller> stereotype.

7. Association.AssociationEnd.name2Controller.Attribute.name: The name of attribute in the «Controller» class is generated from the name of the association end to which is related.
8. Association.AssociationEnd.Class.name2Controller.Attribute.type: The type of the attribute is generated from the name of the class at the association end.
9. Association.AssociationEnd.Class.Method2Controller.Method: The methods of the controller class are generated from the methods of the two classes of the association, provided that these methods relate to the two classes connected by the association that generates the controller.
10. SSD2View: Each system sequence diagram (Generated for each use case element based on SBVR in the CIM level) is transformed into a view class in the PSM level.
11. SSD.name2View.name: The name of the view class is the name of the SSD preceded by the «View» stereotype.
12. AssociationEnd2Contoller-ModelAssociation: The association between the controller and the model is generated from the association end between the class transformed into a model and an association transformed into a controller.
13. SSD.lifeline2Model-ViewAssociation: The association between the model and the view is generated from the membership between the SSD and the lifeline.
14. Class.Method.name2Association.verb: The common method name between the model and the controller to which it is linked, is transformed to a verb of the association.
15. Class(Model | Controller | View).name2AssociationEnd.name: The name of the association end is the name of the class (Model | Controller | View) at the end.
16. AssociationEnd.upper2AssociationEnd.upper/AssociationEnd.lower2AssociationEnd.lower: The upper and the lower cardinality is the same in the corresponding side of the association.

- **BCD&SSD2MVC_DSD**: The business class diagram and system sequence diagram to detailed sequence diagram is a set of transformation rules that allow the ability to automatically generate the detailed sequence diagram of the PSM level from the BCD and SSD of the PIM level. The following rules are applied to each system sequence diagram and for its corresponding classes from the class diagram in the PIM level (Table 2).

1. ActorLifeLine2ActorLifeLine: The Actor in the DSD in the PSM level is generated from the Actor in the SSD of the PIM level.
2. SequenceDiagramSystem2ViewLifeLine: The sequence diagram system is turned into a life line view in the PSM level.
3. Class2ModelLifeLine: Each class that was generated from a DataObject element is turned into a life line model in the DSD of PSM level.
4. Opt2Opt: The option fragment is the same option fragment in the corresponding sequence diagram.
5. Ref2Ref: Reference fragment is generated from the reference fragment in SSD.
6. Alt2Alt: Alt fragment is generated from the alt fragment in SSD.
7. Message2Message:
   (a) Messages between the ActorLifeLine and the ViewLifeLine in SSD are generated from messages exchanged between the Actor and the system in SSD.
   (b) Messages between the ViewLifeLine and ControllerLifeLine are the forward of messages between Actor and ViewLifeLine.
   (c) Messages between ControllerLifeLine and ModelLifeLine are the forward of messages between ViewLifeLine and ControllerLifeLine.
8. Operation2ControllerLifeLine: ControllerLifeLine is generated from a class operation.

Table 1. Transformation rules to generate the detailed class diagram (DCD) from the platform independent model (PIM) level.

N	Source	Target	Rule
1	Class	Model	Class2Model
2	Class.name	Model.name	Class.name2Model.name
3	Class.Attribute	Model.Attribute	Class.Attribute2Model.Attribute
4	Class.Method	Model.Method	Class.Method2Model.Method
5	Association	Controller	Association2Controller
6	Association.verb	Controller.name	Association.verb2Controller.name
7	Association.AssociationEnd.name	Controller.Attribute.name	Association.AssociationEnd.name2Controller.Attribute.name
8	Association.AssociationEnd.Class.name	Controller.Attribute.type	Association.AssociationEnd.Class.name2Controller.Attribute.type
9	Association.AssociationEnd.Class.Method	Controller.Method	Association.AssociationEnd.Class.Method2Controller.Method
10	SSD	View	SSD2View
11	SSD.name	View.name	SSD.name2View.name
12	AssociationEnd	Contoller-ModelAssociation	AssociationEnd2Contoller-ModelAssociation
13	SSD.lifeline	Model-ViewAssociation	SSD.lifeline2Model-ViewAssociation
14	Class.Method.name	Association.verb	Class.Method.name2Association.verb
15	Class(Model | Controller | View).name	AssociationEnd.name	Class(Model | Controller | View).name2AssociationEnd.name
16	AssociationEnd.upper/ AssociationEnd.lower	AssociationEnd.upper/ AssociationEnd.lower	AssociationEnd.upper2AssociationEnd.upper/ AssociationEnd.lower2AssociationEnd.lower

Table 2. Transformation rules to generate detailed sequence diagram (DSD) from the platform specific model (PSM) level.

N	Source	Target	Rule
1	ActorLifeLine	ActorLifeLine	ActorLifeLine2ActorLifeLine
2	SequenceDiagramSystem	ViewLifeLine	SequenceDiagramSystem2ViewLifeLine
3	Class	ModelLifeLine	Class2ModelLifeLine
4	Opt	Opt	Opt2Opt
5	Ref	Ref	Ref2Ref
6	Alt	Alt	Alt2Alt
7	Message	Message	Message2Message
8	Operation	ControllerLifeLine	Operation2ControllerLifeLine

2.3.2. PSM to Code Transformations

The second part of our approach consists of transforming the PSM to code. In this work, we focus on the Java Enterprise Edition application using the three-tiered architecture MVC.

To automate this approach, we define the model-to-text transformations that we apply to DSD and the DCD of the PSM level in order to generate source code (Table 3).

1. Stereotype2Package: In our approach, we have three class stereotypes (Model, View, Controller) from which we generate three packages:
   (a) The name of each package is the combination of the name of the project and the stereotype name.
   (b) Each class is allocated to the package, and generated from the corresponding stereotype.
2. Controller2ServletClass: Servlet classes are generated from classes with the controller stereotype. The Servlet class name is the name generated of the class + "extends httpServlet".
3. Model2ModelClass: A class with the "Model" stereotype is turned into a "JavaBean" class.
4. View2webPage: A class with the "view" stereotype is turned into a "jsp" page.
5. Attribute2Attribute: Each attribute in a class is turned into an attribute of the corresponding generated class.
6. LifeLineController2Operation: A LifeLineController element of DSD is turned into an operation in the Servlet class.
7. Alt2IfCondition: Each "Alt" fragment of DSD is transformed into an "If condition" code part.
   (a) Conditions are generated from the Alt condition in DSD.
   (b) Operations of each condition are generated from an action (Messages, Fragments) inside the "Alt" fragment.
8. Break2Break: "Break" fragment is turned into a break in code.
9. Ref2Operation: "Ref" fragment in DSD is turned into an operation in the Servlet class generated from the corresponding controller. If a "Ref" fragment is inside an "Alt" fragment, the operation is placed in the "If condition" generated from the corresponding "Alt" fragment.
10. Opt2IfCondition: "Opt" fragment is turned, like the "Alt", into an "If" condition in the code. Messages inside the "Opt" fragment are turned into operations of the corresponding "If" condition.
11. Loop2ForLoop: The "For Loop" is generated from the "Loop" fragment in DSD. The "Min" and "Max" values of the loop are generated from "MinInt" and "MaxInt" values specified in DSD.

Table 3. Transformation rules to generate source code from the PSM level.

N	Source	Target	Rule
1	Stereotype	Package	Stereotype2Package
2	Controller	ServletClass	Controller2ServletClass
3	Model	ModelClass	Model2ModelClass
4	View	webPage	View2webPage
5	Attribute	Attribute	Attribute2Attribute
6	LifeLineController	Operation	LifeLineController2Operation
7	Alt	IfCondition	Alt2IfCondition
8	Break	Break	Break2Break
9	Ref	Operation	Ref2Operation
10	Opt	IfCondition	Opt2IfCondition
11	Loop	ForLoop	Loop2ForLoop

3. Implementation of Our Approach

To automate the defined transformation rules in our approach, we need tools that allow for creating input elements (Unified Modeling Language (UML) diagrams), and tools that support the generation of output elements (UML diagrams for the PSM level and source code).

After analyzing and testing existing tools, we decided to use the Eclipse platform with the different existing plug-ins needed to implement our approach, namely the 'Papyrus Modeling' plug-in that supports all UML diagrams elements, QVT, and Acceleo languages for transformations. Besides, we chose to implement our supporting tool, MoDAr-WA, as an Eclipse plug-in that implements the transformations from CIM to PIM then to PSM and finally to source code. Implementing our approach as an Eclipse plug-in, is in fact, done in order to facilitate the use of our transformation approach by designers and developers, and also to benefit from existing plug-ins in Eclipse that we present in following sections.

3.1. Papyrus Modeling Tool

Papyrus is an open source UML tool based on Eclipse. It was developed by the Laboratory of Model-Driven Engineering and Embedded Systems and can be used as a standalone tool or as an Eclipse plug-in. Papyrus is designed to be easily extensible because it is based on a UML profile [8]. In our approach, we use Papyrus as an Eclipse plug-in to model the PIM and PSM levels in this work and the CIM level in our previous works presented in [6].

3.2. Query View Transformation Language

To implement our transformation rules, we have to use a transformation language; there exist many model transformation languages, but QVT is the unique proposal of the OMG. The QVT standard defines three model transformation languages: QVT Operational, QVT Relational, and QVT Core. We decided to use the QVT Operational language because it supports bidirectional transformations, both horizontal and vertical transformations, and ensures automatic traceability [9].

3.3. Acceleo Plug-In

Acceleo is an Eclipse plug-in that implements the model driven architecture (MDA) approach to generate applications source code from EMF-based models. This is an implementation of the Object Management Group (OMG) standard for model-to-text (M2T) transformations [10].

3.4. MoDAr-WA Plug-In

MoDAr-WA is a plug-in based on our defined model-driven development process that allows the ability for automatic code generation for a web application through a set of transformations that take,

as input, the CIM level models in order to generate models in other levels until getting the JavaEE source code that respects the MVC architecture, in this first version of the plugin.

Figure 6 describes the "Transformation" menu item added by the MoDAr-WA plug-in that contains sub-items for each transformation in the approach. This plug-in can be used in two different ways: by executing a general transformation rule to directly generate the JEE (Java Enterprise Edition) project that contains the source code of the application from CIM, or by executing transformation rules separately to generate the following MDA levels in separated folders "PIM" and "PSM", and finally the source code project.

The separated transformation rules give more flexibility to the designer to add, modify, or delete elements in the generated models if necessary.

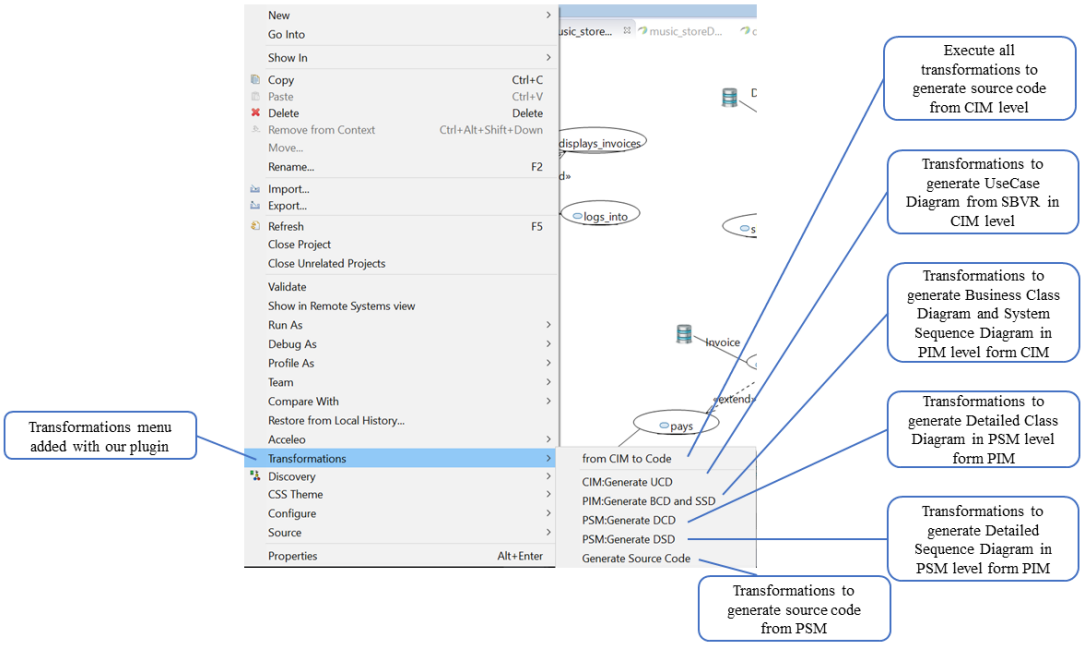

Figure 6. Transformation menu item of the model-driven architecture for web application (MoDAr-WA) plug-in.

4. Case Study

The proposed approach was entirely applied on different projects, such as the library management system and rental car agency system that were partially detailed in [6]. In this paper, we focus on one of the music store systems provided in github [11], as one of the best ways to improve MoDAr-WA is to study and compare applications that have been developed by others, with those generated using our plug-in.

This system is an e-commerce application dealing with the download of sound file samples and albums, developed with Java EE, and respecting the MVC design pattern.

Using the music store system, customers (after subscription) can download or listen to the sound files that are available. They can also add any album to their shopping cart, manage their cart, and even buy the items in this cart. Concerning the system admin, they can process invoices and display downloads.

Figures 7–10 show the music store system defined using SBVR, and the automatic result obtained in each MDA level after applying the implemented transformations.

The process of our approach as previously described starts with the definition of the system features in the CIM level. Figure 7 shows a part of the business vocabulary (BV) and business rules (BR) of the music store system as well as the use case diagram generated through the horizontal transformation of SBVR.

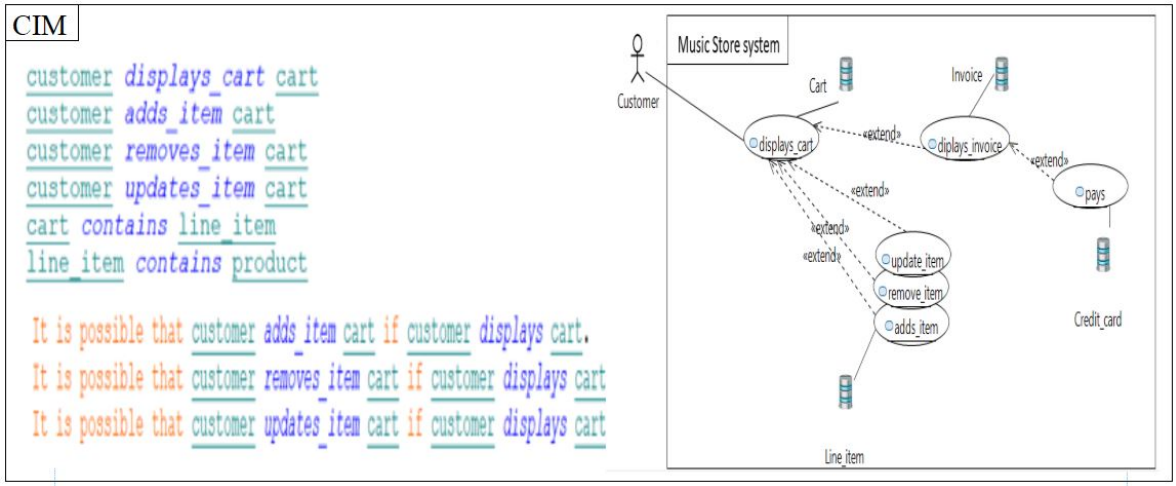

Figure 7. MoDAr-WA process and result of the music store system.

Either by executing the general transformation, or the specific one "CIM to PIM", in MoDAr-WA, we get the PIM level models partially represented in Figure 8. This figure shows the business class diagram and the system sequence diagrams for the "Displays_cart" use case that were automatically generated from the CIM level.

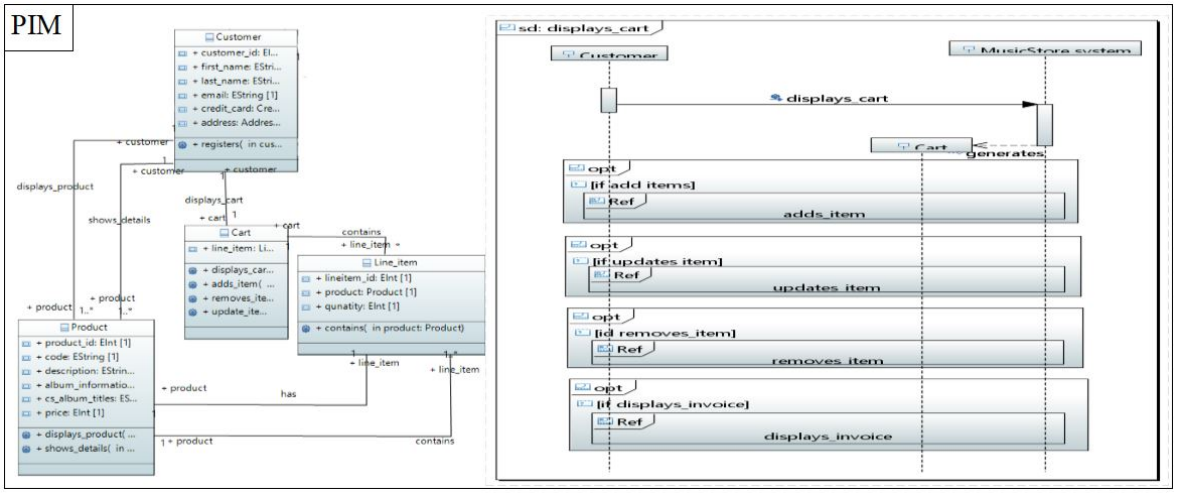

Figure 8. MoDAr_WA process and result of the music store system.

As presented in Figure 9, the generated models in the PSM level are detailed class diagram and detailed sequence diagrams that respect the MVC architecture for web applications.

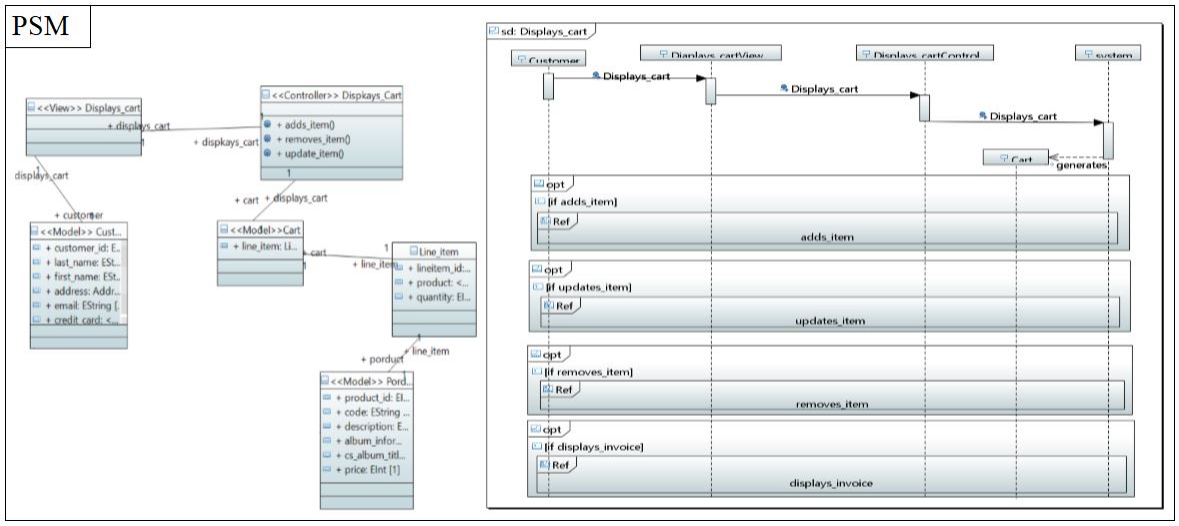

Figure 9. MoDAr-WA process and result of the music store system.

In the end, Figure 10 represents the structure of the generated project and a part of the generated source code.

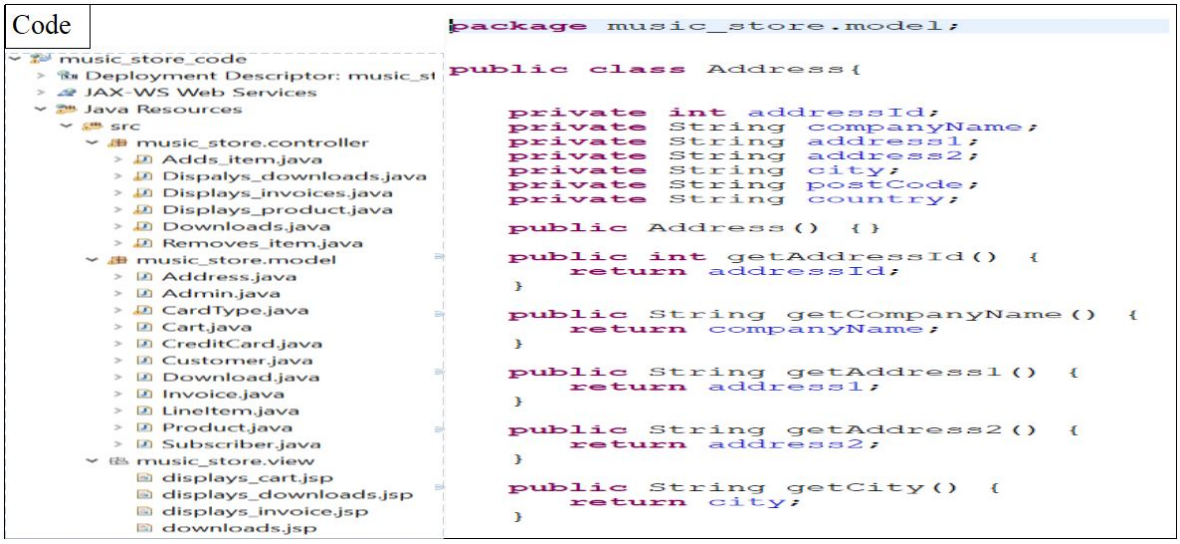

Figure 10. MoDAr_WA process and result of the Music store system.

The package structure, generated code, and all the diagrams generated in the different levels were produced in a few seconds (about 12 s).

This project was very interesting because it was developed by an external team, so it allows us to obtain an objective and constructive comparison to evaluate and improve our tool MoDAr-WA.

Table 4 shows the defined criteria of the comparison between the generated code in MoDAr-WA and the total application code for each MVC layer. The percentage of the generated MVC code elements with respect to the total application code developed by the external team can also be seen in Table 4.

Table 4. Summary of comparison results between the generated MVC code and the total application code.

	Comparison Elements	Generated Code with MoDAr-WA	Total Code	Percentage (%)
Model	Number of Classes	11	10	>100
	Number of Attributes	38	38	100
	Number of Methods(signature)	91	105	86.66
View	Number of views	15	22	68.18
Controller	Number of Controllers	15	5	>100
	Number of Attributes	18	18	100
	Number of Methods	18	32	56.25

The selection of comparison criteria was done according to elements that describe the MVC layers. In the model layer, we made the comparison according to three criteria:

- Number of classes: For this criterion, we checked if MoDAr-WA generated the same classes as defined in the source code application, and that the percentage of this criteria is more than 100%, that is, we generate more classes than was implemented in [11]. This result is due to the traceability ensured by our approach. Indeed, we have defined at the CIM level an "Admin" actor, for example, that was transformed as a model class in the code level, while in the existing code, developers did not implement a model class for this actor but only its views and controllers.
- Number of Attributes: We calculated the total number of attributes of all classes in this layer. For this criterion, the MoDAr-WA plug-in had allowed the generation of 100% of the defined attributes for all model classes. This value is calculated from the comparison of similar classes only in the generated code by MoDAr-WA and the implemented one.
- Number of Methods (with signature and body): For this criterion, we calculated the number of all methods and their signatures in model classes, including "setters" and "getters". As described in Table 4, our approach generated 86.66% of methods of the model classes. Making the comparison between generated methods and manually developed ones, we note that in the existing code, developers have defined methods that detail other ones, while in our approach we generated only the methods defined at the first level CIM.

For the view layer, we defined only one criterion, which is the number of views necessary in order to check if our plug-in has generated the same views as in the existing application code. The percentage of the generated views using MoDAr-WA plug-in was 68.18%. Indeed, in our approach all views are generated from associative fact types that correspond to each actor's features, while in the existing code, other views were defined that are not features.

Finally for the controller layer, we defined three criteria according to which we made the comparison:

- Number of controllers: As in the model layer, we checked if the utilization of the MoDAr-WA plug-in allows the ability to generate controllers as defined in the coded application. The percentage of this criteria was more than 100% and this value could be explained by the fact that, in our approach, we generate for each feature a separated controller, while in the project code, developers used a macro controller where features were defined as methods of the controller.
- Number of Attributes: This defines the total number of attributes in controller classes. For this criterion, our approach has generated 100% of the attributes that are the instantiation of model classes connected by the controller.
- Number of Methods: Calculates the total number of methods in controller classes. As previously mentioned, in our approach we generate, for each feature, a controller, while in the existing code, features are methods in the macro controller, which explain the 56.25% as a value of this criterion.

Out of the previous criteria that describe the layer elements, we mention that our approach allows the generation of all required importations. For example, in the controller classes, MoDAr-WA generated all importations that concern Servlets and exceptions.

5. Related Works

Model driven software development is an important area in software engineering, which is why a wide range of techniques, methods, tools, and approaches have emerged to facilitate software development automation. In this section of the paper, we present a survey and an analysis of tools for automated software development and automatic code generation in order to assess them, compare them to our approach, and determine whether they meet the same objectives as the model driven paradigm. Diverse criteria were considered in this comparison, such as MDA level coverage, code generation, completeness, traceability, and automation of transformation.

To carry out this comparison, we followed a methodology that is composed of three stages. The first stage presents a review of related works to automatic code generation in several academic electronic databases, whereas the second stage presents the evaluation of these works according to the previous criteria. Then, the third and last stage of the methodology includes the report of a comprehensive literature review that identifies technologies, tools, and frameworks in reviewed papers and tools' websites. It is worth mentioning that the selected works and tools are the most relevant from a large set of tools' websites and papers reviewed from the major academic journals, workshops, and international conference proceedings.

Following from this, we present the main features of each approach as well as a summary of advantages and disadvantages, in addition to references for more details about the meta-models and transformation rules used in each method. In examining the current state-of-the-art of the problem under assessment, we studied and compared different approaches of code generation from abstract models following the MDA process, and could state that "most of the existing works propose a code generation approach from the lowest level of abstraction".

For many academic approaches that are an evolution of classic approaches, such as the object oriented hypermedia design method (OOHDM) and hypermedia design method (HDM), consider that UML is the starting point.

OOHDM was originally proposed in 1995 [12] as one of the most important methodologies that aims to separate the design of web systems into three models: conceptual models, navigational models, and abstract interface models. Several proposed approaches are based on OOHDM, as OOHDMDA (MDA for OOHDM) which is an MDE approach [13,14] that uses the PIM-XMI files as an input and generates Servlet-based PSM-XMI files. Thus, the approach defines some PIM-to-PSM transformations that start with OOHDM and end up with Servlets. Although it is based on MDE, it does not follow the entire MDA process, as the approach does not define the CIM level. Furthermore, there are no transformation rules defined to generate source code for the web application.

Ceri et al. defined in their work [4] a WebML method as a notation to specify the conceptual design of complex websites. Different tools were proposed that implements the WebML approach in order to generate source code. WebRatio [15] is one of the most important tools that apply a WebML technique that starts with the conceptual data modeling of the system, then continues with the definition of hypertext models, and then finally the presentation model. Thus, the main advantages of WebRatio are that it implements the entire web development process. However, the authors do not define how to cover all aspects of the MDA level. Indeed, this tool is based on only the UML class diagram that covers only the structural and static aspects of the system while the behavioral or the dynamic ones are not supported. Furthermore, the tool does not ensure any traceability between MDA levels.

As WebML does not define any formal meta-models or model transformations to generate source code from the PIM level of MDA, two alternative approaches appeared; WebML1 and WebML2. WebML1 [16] is a MOF (Meta Object Facility) approach that mainly focuses on defining a meta-model for WebML; transformations are not proposed in this approach. The second approach,

called WebML2, was introduced by Schauerhuber et al. [17] and presents a semi-automatic approach that allows the generation of MOF-based meta-models from document type definition (DTD).

UML-based web engineering (UWE) defines a development process that is able to semi-automatically generate web applications. To achieve this goal, UWE provides a navigation and a presentation model. To create UWE models, an ArgoUWE [18] editor based on an open source tool (ArgoUML) is provided. The transformation code process generates an XML file for deploying a web application into an XML publishing framework. However, the transformation process is currently in a very early development stage, and does not yet generate a complete web application.

A similar approach to UWE is the OOH method that was successfully used for the development of industrial web applications [19]. Indeed, this method is based on three views to model web applications: a class diagram, a navigation diagram, and a presentation diagram. This tool allows for the design of web pages and has a transformation engine that can generate PHP applications [20].

Despite the fact that all previously cited tools allow for the generation of code sources for web applications from models, they do not follow the whole MDA approach and its aspects.

Another example of a model-driven industrial tool is OptimalJ [21]. This tool follows an MDA approach which starts in a domain model (PIM) that is UML compliant. This domain model is transformed into an application model in PSM, from which the final code is generated. Even though OptimalJ is a model-driven tool, its PIM is clearly based on the Java EE platform and is related to technological concepts.

Deeba et al. present in their paper [22] an approach that aims to generate source code for a multi-level marketing scenario from UML diagrams using AndroMDA which is "an extensible generator framework that adheres to the model driven architecture (MDA) paradigm" [5]. In fact, AndroMDA establishes a development methodology to generate deployable components for several platforms and technologies with a definitive goal: write less code. In this work [22], the authors propose an approach that considers the PIM as the highest level of abstraction which is transformed into a generic PSM model for Java EE platform from which different versions of code are generated (Java, C#, and C++). Furthermore, the authors do not describe how to model each level used in the approach nor the transformation rules used.

Esbai et al. propose, in their paper [23], an approach based on MDA and tend to generate from the UML class diagram, the source code files (XML files, JSP pages, and forms) based on an MVC2 pattern. However, this approach does not consider the CIM nor the PIM levels of the MDA approach, indeed the authors proposed an approach that generates code from the PSM level. The PSM level is modeled by a class diagram and transformation rules are defined using QVT which is mainly used for M2M transformations and it is not dedicated to generate source code from models.

Brambilla et al. present in their paper [24], an MDE approach that aims to execute the semantics of BPMN. This approach consists on transforming the manually expressed BPMN models, that describe the business model of the application, to the WebML notation that covers the structure, behavior, and user interaction of the software application, then the executable application running code. However, even if the approach covers the important aspects of the application, the authors do not define any MDA levels and how each one should be modeled in order to respect the abstraction required then the traceability from requirements to source code.

In their paper [25], Hernandez-Mendez et al. present a semi-automatic approach based on MDA to generate a RESTfull web services for single page applications (SPA). This approach starts by the manual definition of the PIM level, which is transformed into a specific model in the PSM level, then generates the code. This approach does not define the CIM level from which the PIM level should be generated. Moreover, the authors did not reveal model aspects at any step of the approach which decreases the traceability and abstraction of the application.

In their paper [26], Dogdu et al. propose a model-driven method to generate elements for web application, which is based on the RDF data model, and partially its reasoning. The approach is implemented as an online front-end framework that takes as an input the data model defined by

developers and generates different views of the data elements automatically. This approach defines the PSM level manually, i.e., it is not generated from the PIM one. Furthermore, the use of RDF (which is a data model that consists on triple statements of subject–predicate–object) is still abstract for the PSM level, as it did not give a description for the technical platform.

In their paper [27], Huang et al. propose an approach that aims to generate source code for web applications from the PSM level modeled by a class diagram using IBM rational rose CASE tool. We note that in this paper, the authors describe transformation rules but do not describe how the mapping into the source code is made.

Bousetta et al. model the PSM level by a state machine diagram and a class diagram that are taken as a source model to which authors apply the EJB3 profile (horizontal transformation) to extend the model by using new features. From the target model results of the previous transformation, they generate the source code of the application. We mention that this work does not detail the transformation rules that allow for generating source code. Moreover, adding the EJB profile in the PSM level is done manually while respecting MDA specifications, this level should be automatically generated from the PIM level [28].

In [29], Albert et al. propose an approach for simplifying the specification of conceptual schemas (CSs). This approach takes as input an UML class diagram and generates an extended version of conceptual schema (CS) with operations that model the system behavior which is transformed into code. Furthermore, this approach aims to automatically generate only a set of basic operations: Create, Update, and Delete. We note that authors aim to generate a behavioral aspect of the system from a structural one. Moreover, the approach does not propose how to model static and dynamic aspects before generating the code.

Bozzon et al. provide, in their paper, an approach that automatically generates code for rich internet applications from conceptual modeling. The paper does not detail transformation rules nor describe how these transformations are implemented [30].

Jamda is an open source framework that tends to generate executable code from the UML domain model which covers the PSM level. The framework performs the role of model compiler in the model driven architecture approach. We mention that generating code from the UML domain model does not cover all PSM level aspects, as it considers only the static aspect of the PSM level [31].

In the book [32], Conallen proposed an approach that models web applications by UML models that are extended with specific elements for web applications using stereotypes, constraints, and tags to define the relationship between a form and a web page. In this book, the author stated that the extended UML diagrams can be transformed to the Java, JSP, and HTML code. However, these transformations are not automated, and the approach of modeling and generating code from models do not follow MDA steps.

In their paper [33], Yilong et al. present an approach RM2PT implemented as a CASE tool for automated MVC code prototype generation from a requirements model in UML diagrams and the formal contracts of their system operations. This approach does not require design models as an input, but relies on a requirements model, which contains: A use case diagram, system sequence diagram, contracts of system operations, and a conceptual class diagram. However, RM2PT does not respect the MDE paradigm, and it does not cover all MDA levels, which means that the traceability of system requirement is not respected in this approach.

After reporting the main approaches proposed to generate web application source code following the model driven architecture process, we present in this section a discussion and a comparison between these approaches and our approach. For this comparison, we defined two criteria according to which we make the comparison:

- Model categories aspects: We discuss if the proposed approaches cover all levels aspects defined, based on OMG specifications.
- Transformation rules: In this part we discuss the automation, completeness, and traceability of transformation rules defined in the different works (N = No, Y = Yes, and P = Partially).

The comparison in this part is based on checking the coverage of MDA levels and their aspects, then the checking of transformation rules that allows the generation of source code and the checking of their automation, completeness, and traceability.

Table 2 summarizes the comparison of the works described above according to these criteria. The first remark that we can make from Table 2 is that the majority of previous works do not cover the CIM level, for example [18,23,28]. Concerning the transformations, we found some works that automate transformations but they ensure neither traceability nor the completeness of rules [26,30]. Other works partially automate transformations and define complete transformation rules [27], however they do not ensure transformation traceability.

In the MoDAr-WA plug-in, each aspect of each MDA level is covered with the appropriate models. The source code according to the requirements modeled in CIM is generated with respect to MVC architecture. Finally, all transformations defined in this approach are automated and ensure completeness and traceability.

Although the previous comparisons presented in this paper have demonstrated the contribution of our approach to this research field, leading a comparative study based on a practical example seems to be necessary to compare obtained results, once using MoDAr-WA, previous approaches, and tools. For that end, we defined three steps that we followed in this analysis. At first, we started by selecting a set of relevant tools based on academic approaches [15,31,33]. Then, in the second step, and in order to expand our analysis, we selected some industrial tools [34,35] that we found complete, and aims to meet the same objectives. Finally, we evaluated the selected tools using the MusicStore example.

A set of criteria, deducted from the combination of previous ones (Tables 4 and 5), was defined to establish the comparison presented in Table 6.

Comparing these approaches to MoDAr-WA, we deduce that:

- MoDAr-WA allows us to generate 86.56% of executable elements from structured system requirements (SBVR).
- MoDAr-WA respects the MDA paradigm. Indeed, it generates J2EE project from CIM, PIM, and PSM levels through successive transformations.
- RM2PT allows us to generate 93.65% of executable elements, which is higher than MoDAr-WA. However, this approach does not respect the MDA paradigm, it covers only PSM level which is manually designed. Thus, a higher time is required to use this approach (as all source elements should be designed by developers). Moreover, this tool does not allow us to generate the complete project, but only separated files containing a generated code.
- The other approaches in the previous table generates less than 50% of source code elements, and does not cover all the MDA levels.

We conclude that our approach presents a real break in the state-of-the-art of automatic code generation from abstract models, following the MDE paradigm. Indeed, MoDAr-WA proposes a new automatic software development process (from the requirements description to the code generation), that ensures traceability and time saving.

Moreover, unlike previous tools, MoDAr-WA presents generic models that could be used for different examples, and does not require the contribution of developers at all steps of the process.

Table 5. Discussion and comparison of previous works.

			Hernandez and al.	Brambilla and al.	OOHDMDA	WebRatio	WebML1	WebML2	UWE (ArgoUWE)	OOH	OptimalJ	Deeba and al.	Esbai and al.	Dogdu and al.	Huang and al.	Boussetta and al.	Albert and al.	Bozzon and al.	Jamda	Conallen
CIM	Covered		N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	N
	Aspects	Functional	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N	N	N	N
		Static	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
		Behavioural	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
PIM	Covered		Y	N	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	N	N	N	N	N
	Aspects	Structural	N	N	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	N	N	N	N	N
		Dynamic	N	N	Y	N	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N
PSM	Covered		Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
	Aspects	Static	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
		Structural	N	N	N	N	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
		Behavioural	N	N	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	Y	N	N	N
		Dynamic	N	N	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N
Code	Generated		Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
Transformations	Completness		N	Y	Y	P	N	P	P	P	P	P	P	N	Y	P	N	N	N	N
	Automation		P	P	P	N	N	P	P	P	Y	P	N	P	P	P	N	P	P	N
	Traceability		N	N	N	N	N	N	N	N	N	N	N	N	P	N	N	N	N	N

Table 6. Obtained comparison results.

Criteria\Approaches		**Telosys**	**Jamda**	**Umple**	**WebRatio**	**RM2PT**	**MoDAr-WA**
Covered levels	CIM	N	N	N	N	N	Y
	PIM	N	N	N	Y	N	Y
	PSM	Y	Y	Y	Y	Y	Y
	Code	Y	Y	Y	Y	Y	Y
Source		Database Model, DSL, Templates	UML Models (XML format)	UML Class Diagram, State Machine	UML Class Diagram	UML UCD, SSD, Contracts of Systems Operations, Conceptual Class Diagram	SBVR, UCD
Target		Java Classes, XML, HTML	J2EE Sample	Java, C++, PHP, Ruby	Java code for web application	MVC Java code	J2EE Executable application project
% Generated elements		<50%	<50%	<50%	<50%	93.65%	89.56%

6. Conclusions and Future Work

Raising the level of abstraction of the execution platforms by modeling requirements through a computation independent model (CIM), model driven architecture approach offers an interesting and original dimension to the software engineering domain [36]. As stated by the Object Management Group, MDA is about using modeling languages as programming languages rather than merely design languages.

In this spirit, we proposed an approach that aims to emphasize the necessity of models in software development. Indeed, our approach follows the MDA principles and proposes a definition of the various aspects for each MDA level and also transformation rules to move from one level to another until the automatic generation of source code for web applications that respect the MVC architecture. In order to automate this approach, we implement it as an Eclipse plug-in; MoDAr-WA is also is based on other plug-ins (EMF, UML, Papyrus).

An e-commerce project was used to compare and validate the benefits behind the use of our approach according to several criteria that was defined for each MVC layer (number of model classes, number of attributes, number of views, etc.). Furthermore, in order to statute our approach among the previous works that aim to generate web application source code, we have performed a comparison to check the completeness, the automation, and the traceability of the proposed transformations.

The main contributions of this work can be summarized as follows:

- An MDA approach that takes into account all levels and aspects of MDA and provides traceability. This approach introduces a set of meta-models to define the relevant information in each MDA level and set of transformations to generate the source code of MVC web applications from models in CIM.
- A tool support MoDAr-WA that allows the automatic generation of web application source code from requirements description in CIM using SBVR. The plug-in MoDAr-WA makes also possible the use of elementary transformation from one level to another separately. Thus, the user could enrich or modify the generated models in one level before generating models or code in the following level.

Several directions are envisaged to further enhance our proposed solutions in the context of modeling and automation of software development life cycle, but also as a part of the optimization of the existing one. Hence, we are looking at:

- Generating a more generic PSM models from PIM: In the current approach, we generate diagrams representing the PSM level that are specific for JavaEE MVC web application, so in our future work we aim to propose a solution where we define a generic meta-model of the PSM level that can support different technical platforms. Then, through an horizontal transformation in the same level a specific PSM for the desired technical platform can be automatically generated to finally generate the application source code.
- Combining model based testing (MBT) with MDA to deal with the testing phase, which is one of the important steps in software development life cycle. As the key of our approach is based on models and follows MDA principles, we have proposed in one of our previous works [37], an approach that allows the generation of different types of test cases from modeling the different MDA levels. In the future, we intend to automate and improve these transformations in order to automatically generate test cases from models.
- Integrating MDA in agile methods or even proposing a new one that deals with MDA. The automation of MDE approaches (MDA and MBT in our case) allows us to ensure the portability, interoperability, and traceability of developed systems. However, the evolution of the system requirement is still an issue to deal with and that constitute a promoting research axis. In order to deal with this issue, we aim to propose an agile approach where models are the key of development. We mention that we presented in our work [38]—our reflection on the combination

of MDA and some agile methodologies and incremental life cycles. We have also proposed an approach that combines MDA, MBT, and V life cycle in scrum sprints [39]. More work and experiments must be done to perform these propositions in a fully automatic way.

Author Contributions: Conceptualization, I.E.; methodology, I.E.; software, I.E.; validation, I.E., S.C. and M.R.; formal analysis, I.E. and S.C.; investigation, I.E.; resources, I.E.; writing—original draft preparation, I.E.; writing—review and editing, I.E. and S.C.; visualization, I.E.; supervision, S.C. and M.R.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MoDAr-WA	Model Driven Architecture for Web Application
MDA	Model Driven Architecture
M2M	Model to Model
M2T	Model to Text
OMG	Object Management Group
UML	Unified Modeling Language
QVT	Query View Transformation
UCD	Use Case Diagram
SBVR	Semantic Business Vocabulary and Business Rules
BCD	Business Class Diagram
SSD	System Sequence Diagram
DCD	Detailed Class Diagram
DSD	Detailed Sequence Diagram
CIM	Computation Independent Model
PIM	Platform Independent Model
PSM	Platform Specific Model
OOH	Object-Oriented Hypermedia
SPA	Single Page Application
OOHDM	Object Oriented Hypermedia Design Method
HDM	Hypermedia Design Method

References

1. Miranda, M.A.; Ribeiro, M.G.; Marques-Neto, H.T.; Song, M.A.J. Domain-specific language for automatic generation of UML models. *IET Softw.* **2017**, *12*, 129–135.
2. Bézivin, J. On the unification power of models. *Softw. Syst. Model.* **2005**, *4*, 171–188.
3. Koch, N.; Kraus, A. The Expressive Power of UML-based Web Engineering. In Proceedings of the Second International Workshop on Web-oriented Software Technology (IWWOST02), Málaga, Spain, 10–14 June 2002.
4. Ceri, S.; Fraternali, P.; Bongio, A. Web Modeling Language (WebML): A modeling language for designing Web sites. *Comput. Netw.* **2000**, *33*, 137–157.
5. AndroMDA. 2008. Available online: http://andromda.org/modeling.html (accessed on 3 December 2019).
6. Essebaa, I.; Chantit, S. Tool Support to Automate Transformations between CIM and PIM Levels. In Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering—Volume 1: MDI4SE, INSTICC, Porto, Portugal, 28–29 April 2017; SciTePress: Setúbal, Portugal, 2017; pp. 367–378, doi:10.5220/0006388703670378.
7. Essebaa, I.; Chantit, S. Tool Support to Automate Transformations from SBVR to UML Use Case Diagram. In Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering—Volume 1: MDI4SE, INSTICC, Madeira, Portugal, 23–24 March 2018; SciTePress: Setúbal, Portugal, 2018; pp. 525–532, doi:10.5220/0006817705250532.
8. Papyrus. 2010. Available online: https://www.eclipse.org/papyrus/resources/PapyrusTutorial_OnSequenceDiagrams_v0.1_d2010100.pdf (accessed on 3 December 2019).

9. MOF. OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. 2009. Available online: http://www.omg.org/spec/QVT/1.0/PDF (accessed on 3 December 2019).
10. Acceleo. 2006. Available online: https://www.eclipse.org/acceleo/ (accessed on 3 December 2019).
11. MusicStore Project. 2015. Available online: https://github.com/MichalGoly/MusicStore (accessed on 3 December 2019).
12. Schwabe, D.; Rossi, G. The object-oriented hypermedia design model. *Commun. ACM* **1995**, *38*, 45–47.
13. Schmid, H.A. Model Driven Architecture with OOHDM. In Proceedings of the the 4th International Conference on Web Engineering, Munich, Germany, 28–30 July 2004; pp. 12–25.
14. Schmid, H.A.; Donnerhak, O. OOHDMDA—An MDA approach for OOHDM. In Proceedings of the International Conference on Web Engineering, Sydney, NSW, Australia, 27–29 July 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 569–574.
15. WebRatio. 2011. Available online: http://www.webratio.com (accessed on 3 December 2019).
16. Moreno, N.; Fraternalli, P.; Vallecillo, A. A UML 2.0 profile for WebML modeling. In Proceedings of the Workshop Sixth International Conference on Web Engineering, Palo Alto, CA, USA, 10–14 July 2006; ACM: New York, NY, USA, 2006; p. 4.
17. Schauerhuber, A.; Wimmer, M.; Kapsammer, E. Bridging existing Web modeling languages to model-driven engineering: A metamodel for WebML. In Proceedings of the Workshop Sixth International Conference on Web Engineering, Palo Alto, CA, USA, 10–14 July 2006; ACM: New York, NY, USA, 2006, p. 5.
18. Knapp, A.; Koch, N.; Zhang, G. Modeling the structure of web applications with argouwe. In Proceedings of the International Conference on Web Engineering, Munich, Germany, 26–30 July 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 615–616.
19. Abrahão, S.; De Marco, L.; Ferrucci, F.; Gomez, J.; Gravino, C.; Sarro, F. Definition and evaluation of a COSMIC measurement procedure for sizing Web applications in a model-driven development environment. *Inf. Softw. Technol.* **2018**, *104*, 144–161.
20. Gómez, J. Model-driven web development with visualwade. In Proceedings of the International Conference on Web Engineering, Munich, Germany, 26–30 July 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 611–612.
21. OptimalJ. 2007. Available online: http://www.compuware.com/products/optimalj (accessed on 3 December 2019).
22. Deeba, F.; Kun, S.; Shaikh, M.; Dharejo, F.A.; Hayat, S.; Suwansrikham, P. Data transformation of UML diagram by using model-driven architecture. In Proceedings of the 2018 IEEE 3rd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), Chengdu, China, 20–22 April 2018; pp. 300–303.
23. Esbai, R.; Erramdani, M.; Mbarki, S.; Arrassen, I.; Meziane, A.; Moussaoui, M. Transformation by modeling MOF 2.0 QVT: from UML to MVC2 web model. *INFOCOMP* **2011**, *10*, 1–11.
24. Brambilla, M.; Fraternali, P. Implementing the semantics of BPMN through model-driven web application generation. In Proceedings of the International Workshop on Business Process Modeling Notation, Lucerne, Switzerland, 21–22 November 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 124–129.
25. Hernandez-Mendez, A.; Scholz, N.; Matthes, F. A Model-driven Approach for Generating RESTful Web Services in Single-Page Applications. In Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2018), Funchal, Madeira, Portugal, 22–24 January 2018; pp. 480–487.
26. Dogdu, E.; Hakimov, S.; Yumusak, S. A data-model-driven web application development framework. In Proceedings of the 2014 ACM Southeast Regional Conference, Kennesaw, Georgia, 28–29 March 2014; ACM: New York, NY, USA, 2014; p. 47.
27. Huang, Y.C.; Chu, C.P.; Lin, Z.A.; Matuschek, M. Transformation from Web PSM to Code. In Proceedings of the International Conference on Distributed Multimedia Systems (DMS), Lisbon, Portugal, 8–9 July 2009.
28. Bousetta, B.; Beggar, O.E.; Gadi, T. A Model Transformation Approach for Code Generation from State Machine Diagram. *IADIS Int. J. Comput. Sci. Inf. Syst.* **2014**, *9*, 1–15.
29. Albert, M.; Cabot, J.; Gómez, C.; Pelechano, V. Generating Operation Specifications from UML Class Diagrams: A Model Transformation Approach. *Data Knowl. Eng.* **2011**, *70*, 365–389. doi:10.1016/j.datak.2011.01.003.
30. Bozzon, A.; Comai, S.; Fraternali, P.; Carughi, G.T. Conceptual Modeling and Code Generation for Rich Internet Applications. In Proceedings of the 6th International Conference on Web Engineering

(ICWE '06), Palo Alto, CA, USA, 11–14 July 2006; ACM: New York, NY, USA, 2006; pp. 353–360, doi:10.1145/1145581.1145649.
31. Jamda. 2013. Available online: http://jamda.sourceforge.net/ (accessed on 3 December 2019).
32. Conallen, J. *Building Web Applications with UML*; Addison-Wesley: Boston, MA, USA,2002.
33. Yang, Y.; Li, X.; Liu, Z.; Ke, W. RM2PT: A Tool for Automated Prototype Generation from Requirements Model. In Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19), Montreal, QC, Canada, 25–31 May 2019; IEEE Press: Piscataway, NJ, USA, 2019; pp. 59–62, doi:10.1109/ICSE-Companion.2019.00038.
34. Telosys. 2018. Available online: https://modeling-languages.com/telosys-tools-the-concept-of-lightweight-model-for-code-generation/ (accessed on 3 December 2019).
35. Umple. 2018. Available online: https://cruise.eecs.uottawa.ca/umple/ (accessed on 3 December 2019).
36. De Lara, J.; Guerra, E.; Cuadrado, J.S. Model-driven engineering with domain-specific meta-modelling languages. *Softw. Syst. Model.* **2015**, *14*, 429–459.
37. Essebaa, I.; Chantit, S. A Combination of V Development Life Cycle and Model-based Testing to Deal with Software System Evolution Issues. In Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development—Volume 1: MODELSWARD, INSTICC, Funchal, Madeira, Portugal, 22–24 January 2018; SciTePress: Setúbal, Portugal, 2018; pp. 528–535, doi:10.5220/0006657805280535.
38. Essebaa, I.; Chantit, S. Model Driven Architecture and Agile Methodologies: Reflexion and discussion of their combination. In Proceedings of the 2018 Federated Conference on Computer Science and Information Systems (FedCSIS 2018), Poznan, Poland, 9–12 September 2018; pp. 939–948, doi:10.15439/2018F358.
39. Essebaa, I.; Chantit, S. Scrum and V Lifecycle Combined with Model-Based Testing and Model Driven Architecture to Deal with Evolutionary System Issues. In *Model and Data Engineering, Proceedings of the 8th International Conference on Model and Data Engineering, Marrakesh, Morocco, 24–26 October 2018*; Springer: Cham, Switzerland, 2018; pp. 77–91, doi:10.1007/978-3-030-00856-7_5.