# Applying a Model Driven Architecture Approach: Transforming CIM to PIM Using UML

https://doi.org/10.3991/ijoe.v14i09.9137

Oualid Betari (✉), Saida Filali, Amine Azzaoui, Mohamed Amine Boubnad
Mohammed First University, Oujda, Morocco
beta.oualid@gmail.com

**Abstract—**Over the last few years, as they evolve with business needs and technology, enterprises are faced with the need to adapt their business processes to work in open settings. In such settings, the automation and the interoperability of business process and applications become a key concern. The Model Driven Architecture (MDA) is introduced as an approach to cope with this challenge. MDA specifies four levels of abstraction, most of the existing studies focus on modeling and transforming the Platform-Independent Model (PIM) to Platform-Specific Model (PSM) levels, while the more conceptual level, the Computation-Independent Model (CIM) is often presumed as present and is not further studied. In this paper, we propose an approach for transforming a CIM into a PIM using the core modeling concepts of the Unified Modeling Language (UML). One important characteristic of this approach is that it provides a method to capture and describe the requirements of the business process using a use cases model. The other important characteristic is proposing an architecture of the PIM based on the classes model. The execution of the transformation is programmed by the Query View Transformation (QVT) language.

**Keywords—**model driven architecture, computation independent model, platform independent model, unified modeling language, use cases, classes

## 1 Introduction

Computing infrastructures are expanding their reach in every dimension. New platforms and applications must interoperate with legacy systems. The Internet, where business relationships exhibit a high degree of dynamism, is imposing new integration challenges as it extends into every corner of every organization [1-2]. These challenges have given rise to the situation of enterprises not only expanding their businesses but also increasing their vulnerability.

In order to handle these changes, enterprises must adapt their business process, one approach to do that is by using the Model driven architecture (MDA). This approach, proposed by the Object Management Group (OMG), supports evolving standards in application domains. MDA provides an open, vendor-neutral approach to the challenge of interoperability, building upon and leveraging the value of OMG's established modeling standards: Unified Modeling Language (UML); Meta-Object Facility (MOF) [1].

MDA is an approach to software development that highlights modeling. MDA provides ways to use models at the different phases of application development. MDA separates the specification of system functionality in a Platform Independent Model (PIM) from the specification of the implementation of that functionality on a specific technology in a Platform Specific Model (PSM). Furthermore, the system requirements are specified in a Computation Independent Model (CIM) [3-4].

Computation independent model (CIM) does not show details of the system structure. CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirement on the other. CIM always uses a vocabulary that is familiar to the practitioners of the domain in question [5].

Platform independent model (PIM) represents the business model to be implemented by an information system. PIM describes processes and structure of the system without reference to the delivery platforms. PIM ignores operating system, programming languages, hardware and networking.

A platform specific model (PSM) combines specifications in the PIM with details that specify how a system uses a particular type of platform.

As these different types of models represent different levels of abstraction of the same system, MDA recommends the use of transformation mechanisms allowing to move from one level to another. It forms a key part of MDA. The OMG proposal for a transformation language is QVT [1].

Considering these aspects, the present paper describes some important cases of transformation from CIM to PIM and propose a new approach by modeling to realize the transformation. In this paper, we discuss some important cases of transformation from CIM to PIM and propose a new approach which transform a set of UML Use Cases, considered to be a CIM, to a set of UML Classes, considered as a PIM.

The rest of this paper is structured as follows. Section 2 presents the most relevant related works. Section 3 presents the background knowledge of the paper, we outline the MDA principles and the concepts of the Unified Modeling Language. In section 4 we shall summarize our MDA approach for transforming CIM to PIM. The transformation rules will be presented in section 5. Finally, section 6 concludes the paper and offers further perspectives.

## 2 Related works

In this section, we present a brief overview of some approaches, proposed for the transformation from computation independent model to platform independent model.

Cao in [5] addressed this problem and proposed an approach for the transformation with pattern. In this approach, they take advantage of "reuse" from various standpoints. Feature model is used to describe the requirement of the application. Moreover they use pattern to transform CIM to PIM.

An analytical CIM to PIM transformation in [6] provides one way to transform business requirements to UML models using Data Flow Diagrams which represent business processes as models for CIM level description. For the PIM level they consider UML diagrams.

Kherraf in his proposition [7] describes a disciplined approach to transform a CIM into a PIM. It first uses UML2 activity diagrams to model the business processes up to the users' tasks. The activity diagrams are then detailed to specify the system requirements. The system components are directly deduced from the requirement model elements.

Zhang [8] presented a feature-oriented component-based approach to the CIM-to-PIM transformation. In his approach, features and components were adopted as the key elements of CIM and PIM, respectively. His approach provided a method to decompose the n -to-n relations between features and components into two groups of 1-to-n relations.

In [9-10-11], they propose an approach where an SBP description corresponds with a CIM model and can be used as a complement to the Business Modeling discipline of the UP. In addition, the Use Cases, which form a part of a PIM model, will complement the Requirement and Analysis & Design disciplines.

The method in [12] aims at providing a solution to the problem of constructing CIM and its automatic transformation at the PIM using the QVT transformation rules. The approach proposes to represent CIM by two models: The business process model and the functional requirement model.

So we present a new approach in which we focus on using UML in the different levels of the transformation process. We capture and describe the requirements of the business process using an use cases model, which represents the CIM, and we use the classes diagram to represent the PIM. Then we provide the transformation rules from CIM to PIM by using the QVT language, to improve the efficiency and the degree of automation.

## 3 Background knowledge

### 3.1 Model driven engineering

In late 2000, OMG, a consortium of over 1,000 companies, first reviewed the document entitled "Model Driven Architecture" and decided to form an architecture team to produce a more formal statement of the MDA [1]. This approach focuses on developing the highest level of abstraction models and promotes the transformation approach from one model to another.

MDA addresses the challenges of today's highly networked, constantly changing systems, providing an architecture that assures portability, cross-platform Interoperability, platform independence, domain specificity and productivity [1]. The key to the MDA approach is the importance of models in the software development process. In the MDA, the software development process is driven by the modeling activity of the software system.

In the field of software engineering, the OMG with its MDA approach classifies four types of models that it advocates for the construction of software: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) and Code; defined as follow:

- CIM: The goal is to create a requirements model for the future application. Such a model must represent the application in its environment in order to define the services offered by the application and which other entities with which it interacts.
- PIM: This model represents the system-specific business logic or the design model. It represents the functioning of entities and services. It must be durable and lasting over time. It describes the system, but does not show the details of its use on the platform.
- PSM: MDA considers that the code of an application can be easily obtained from these models. The main difference between a code model and an analysis or design model is that the code model is linked to an execution platform.

The reason for the above model organization is to develop models of the systems' business logic independently from the platforms of execution, then to transform these models automatically to models dependent of the platforms. The complexity of the platforms does no longer appear in the business logic models but it' is found in the transformation [3].

The required steps during the model-driven development with the UML approach can basically be divided into the following steps [4], at first building the CIM that acquires user requirements. Then, according to this CIM, a PIM is built. Next is, the transformation of the proposed PIM into one or more PSMs. This type of transition from CIM to PIM and PIM to PSM is called Model To Model (M2M) transformation. The final step is to transform the generated model respecting the PSM into the code of the chosen platform. This transition is called Model To Text (M2T) transformation. Figure 1, shows how the transformations are done.

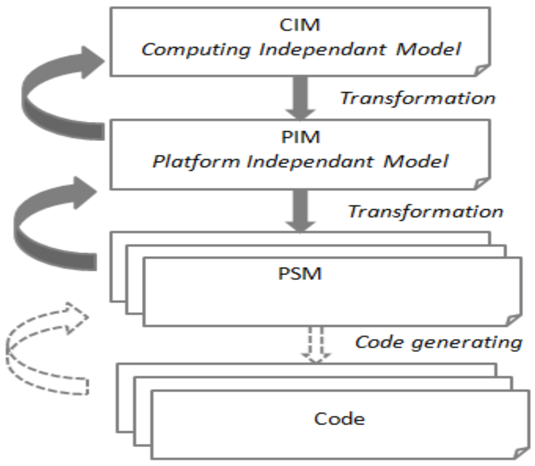

**Fig. 1.** Model Driven Architecture levels

The OMG, in the context of MDA, gives the following definition to the transformation of models "the process of converting a model into another model of the same system". In general, it can be said that a transformation definition consists of a collection of transformation rules, which are unambiguous specifications of how a model can be used to create another model. Using the modeling approach is designed to have a sustainable and productive models' transformation, independently of any execution platform. This is why the OMG has developed a standard for this transformation language which is the MOF 2.0 QVT [13], standing for Query View Transformation.

### 3.2 UML models

The use of models in the design of complex engineering systems is a long-standing tradition that is almost as old as engineering. Yet, its applicability to software has often been questioned. Modeling and model-based techniques are, in fact, the only viable way of coping with the kind of complexity that is encountered in modern software systems [14].

Our approach is a fully automatic way of transformation from CIM to PIM and it provides one way to transform business requirements to software models. At first we had to determine which models represent CIM level and which models represent PIM level of MDA. For this models, we used UML, which is a way of visualizing a software program using a collection of diagrams. In the next chapter, we'll be introducing those models.

**Models representing our CIM level (use case diagram) :** The use case is a popular representation of requirements that describes a set of scenarios. It was originally created as part of the Objectory process for object-oriented software development.

A use case is defined as "the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system" [15-16].

**Models representing our PIM level (classes diagram) :** A class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations. Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes [17].

## 4 CIM to PIM transformation

In this paper we proposed an use cases diagram as a CIM. This model will be transformed into a UML classes diagram, representing our PIM, with an approach by modeling using QVT. This type of transformation will allow us to automatically generate the classes from the systems' requirements. We used UML in our approach, because it takes into account the structural and dynamic properties of the information system at a conceptual level.

### 4.1 CIM source meta model

The first thing to do when building a new application is of course to specify the requirements of the client. Although very upstream, this step should benefit greatly from the models. The goal is to create a requirements model for the future application. Such a model must represent the application in its environment in order to define the services offered by the application and other entities with which it interacts.

Use cases are a means for specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is why we opted for a UML use cases, represented as an MOF class diagram, as our CIM. In Figure 2, we showcase the metamodel and the meta-classes representing the key concepts of use cases [15].

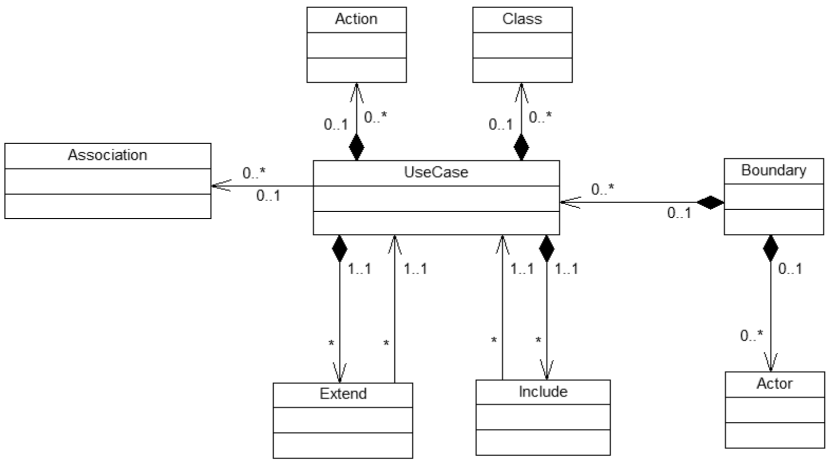

**Fig. 2.** Simplified Metamodel of use cases diagram

- *Actor* : An actor is behaviored classifier which specifies a role played by a user or any other system that interacts with the subject. An Actor models a type of role played by an entity that interacts with the subject, but which is external to the subject. Since an actor is external to the subject, it is typically defined in the same classifier or package that incorporates the subject classifier.
- *Extend* : This relationship specifies that the behavior of a use case may be extended by the behavior of another.
- *Include* : Include is a directed relationship between two use cases, implying that the behavior of the included use case is inserted into the behavior of the including use case.
- *UseCase* : represents a use case, enabling more than one of them to be specified in a single model in conformance to this metamodel. A UseCase is a kind of behaviored classifier that represents a declaration of an offered behavior. Each use case specifies some behavior, possibly including variants, that the subject can perform in collaboration with one or more actors. Each use case is divided into two terms. The first one describes the *action* part performed by the user, while the second characterizes the *class* representing a category of objects.

### 4.2 PIM source Metamodel

The PIM represents the system-specific business logic or the design model. It represents the functioning of entities and services. It must be durable and lasting over time. It describes the system, but does not show the details of its use on the platform. At this level, the formalism used to express a PIM is a class diagram in UML, represented in Figure 3.

The Classes package contains sub packages that deal with the basic modeling concepts of UML, and in particular classes and their relationships. The Kernel package represents the core modeling concepts of the UML, including classes, associations, and packages. Our source Metamodel structures a simplified UML model based on a package containing the data types and classes. The classes contain structural features represented by attributes, and behavioral features represented by operations.

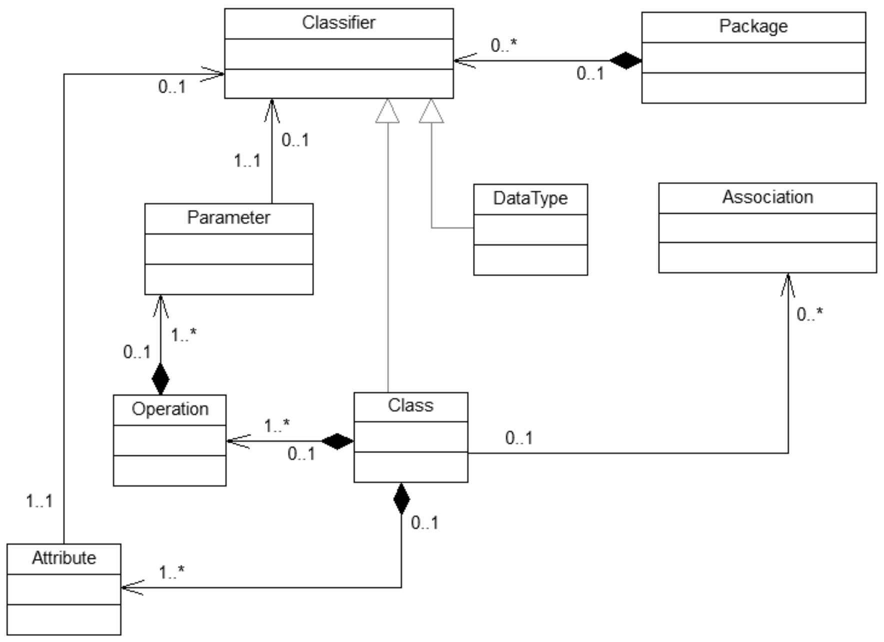

**Fig. 3.** Simplified Metamodel of classes diagram

- *Association* : An association specifies a semantic relationship that can occur between typed instances. It has at least two ends represented by properties, each of which is connected to the type of the end. More than one end of the association may have the same type.
- *Package* : A package is used to group elements, and provides a namespace for the grouped elements.
- *Classifier* : A classifier is a namespace whose members can include features. Classifier is an abstract metaclass which describes set of instances having common fea-

tures. This meta-class represents both the concept of class and the concept of data type.

- *Class* : Class is a kind of classifier whose features are attributes and operations. Attributes of a class are represented by instances of Property that are owned by the class.
- *DataType* : A data type is a type whose instances are identified only by their value. A DataType may contain attributes to support the modeling of structured data types.
- *Parameter* : A parameter is a specification of an argument used to pass information into or out of an invocation of a behavioral feature.

## 5 The transformation process

Once the meta models developed, the next step is to specify the correspondences between the two meta models by using transformation rules written in QVT Operational Mappings. In this section we'll explain some of the programmed transformation rules. In Table 1, rules expressed in textual QVT are described.

**Table 1.** Mapping between Use Case elements and Classes

| **Transformation rule** | **Source** | **Target** | **Description** |
|---|---|---|---|
| *ActorsToClass* | Actor | Class | This transformation will map the actor from the use case diagram into a class the classes diagram. |
| *UseCaseToClass* | Class | Class | The class part of a use case will be transformed into a class. |
| | Action | Operation | This will map the action part of a use case into the class' operation. |
| *AssociationToAssociation* | Association | Association | Maps the association between use cases into an association between classes. |
| *IncludeToAssociation* | Include | Association | Transform the include relationship into an association between classes. |
| *ExtendToGeneralization* | Extend | Generalization | Map the extend relation between two use cases into a relation of generalization. |
| *BoundaryToPackage* | Boundary | Package | Transforms the use case boundary to a package. |

We first developed our models corresponding to our source and target metamodels, and then we implemented the algorithm (see below) using the transformation language QVT Operational Mappings.

**Main Algorithm :**

```
  input srcModel:UCDBoudary
  output destModel:CDPackage
  begin
  create UCDBoundary UCD
  create CDPackage CD
  for all a in srcModel
  map UCDBoudaryToCDPackage (a)
  end for
  end
  mapping UCDBoudaryToCDPackage (ucd:UCD.UCDBoudary):
CD.CDPackage
  begin
  create CDPackage cdp
  cdp.name = ucd.name + ‘CD’
  for all cdp.class
    map actorToClass()
    map usecaseToClass()
  end for
  for all cdp.association
    map associationToAssociation()
  end for
  end
  mapping actorToClass (ac:Actor):Class
  begin
  create Class c
  c.name = ac.name + ‘Class’
  end
  mapping usecaseToClass (uc:UseCase):Class
  begin
  create Class c
  c.name = uc.class.name + ‘Class’
  for all c.operation
    map actionToOperation()
  end for
  end
  mapping actionToOperation (at: Action):Operation
  begin
  create Operation o
  o.name = at.name + ‘Op’
  end
  mapping associationToAssociation
(a:Association):Association
  begin
  create Association ac
  ac.name = a.name + 'AC'
  for all ac.generalization
    map extendToGeneralization ()
  end for
  end
  mapping extendToGeneralization (et: Ex-
tend):Generalization
  begin
  create Generalization g
  g.name = et.name + 'Et'
  end
  end
```

**QVT transformations :** A QVT transformation is in the form of a function with two parameters, the first corresponds to the source model, the second corresponds to the target model. The main function of our transformations is as follow:

```
  transformation ucdToCdTransfo(in src:UCD, out dest:CD);
  main() {
    src.objectsOfType(UcdBoundary) -> map ucdBoundToCd-
Pack();
  }
```

We consider that a simplified use case model consists of a Boundary called UcdBoundary. It will be transformed into a class' package called CDPackage, using the transformation rule defined as follows:

```
  mapping UCD::UcdBoundary:: ucdBoundToCdPack() :
CD::CDPackage {
    result.name := self.name + "Cd";
    result.association += self.actor[UCD::Association] ->
map associationToAssociation();
  }
```

The following code shows the transformation rule of an association from a use case diagram to an association between classes.

```
  mapping Association::associationToAssociation () : As-
sociation {
    result.name := self.name + "Ac";
    generalization += self.extend -> map extendToGeneral-
ization();}
```

## 6 Conclusion

This paper presents a new approach of CIM-to-PIM transformation, in which, we focus on the software development from the UML angle. In our approach, the UML diagrams are used as CIM and PIM. This approach introduces the use cases diagram as the model of the CIM level, which will be transformed into classes diagrams modeling the PIM level.

The primary benefit of our approach is that it provides a disciplined way towards automation of the CIM-to-PIM transformation.

Our future intentions include fortifying our MDA approach by the use of the UML sequences and activities diagrams. Also, we aim to apply the UML approach for the other levels transformations, such as PIM to PSM.

## 7 References

[1] Object Management Group. MDA GuideV2.0. (2014-06-01). http://www.omg.org/mda/.

[2] Klaus, F., Jörg, M., & Renato, L. (2012). Agent-Based Technologies and Applications for Enterprise Interoperability. Springer-Verlag Berlin Heidelberg.

[3] Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). MDA Explained: the Model Driven Architecture: Practice and Promise. Addison-Wesley Professional.

[4] Mellor, S., Scott, K., Uhl, A., & Weise, D. (2004). MDA Distilled: Principles of Model-driven Architecture. Addison-Wesley.

[5] Cao, X. X., Miao, H. K., & Chen, Y. H. (2008). Transformation From Computation Independent Model to Platform Independent Model with Pattern. Journal of Shanghai University, 12(6):515-523. https://doi.org/10.1007/s11741-008-0610-2

[6] Kardoš, M., & Drozdová, M. (2010). Analytical Method of CIM to PIM Transformation in Model Driven Architecture (MDA). Journal of Information and Organizational Sciences, 34(1):89-99.

[7] Kherraf, S., Lefebvre, É., & Suryn, W. (2008). Transformation From CIM to PIM Using Patterns and Archetypes. 19th Australian Conference on Software Engineering (ASWEC). IEEE. https://doi.org/10.1109/ASWEC.2008.4483222

[8] Zhang, W., Mei, H., Zhao, H., & Yang, J. (2005). Transformation From CIM to PIM: A Feature-Oriented Component-Based Approach. International Conference on Model Driven Engineering Languages and Systems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11557432_18

[9] Rodríguez, A., Fernández-Medina, E., & Piattini, M. (2007). Towards CIM to PIM Transformation: From Secure Business Processes Defined in BPMN to Use-Cases. International Conference on Business Process Management. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-75183-0_30

[10] Rodríguez, A., Fernández-Medina, E., & Piattini, M. (2008). CIM to PIM Transformation: A reality. Research and Practical Issues of Enterprise Information Systems II. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-76312-5_50

[11] Rodríguez, A., de Guzmán, I. G. R., Fernández-Medina, E., & Piattini, M. (2010). Semi-Formal Transformation of Secure Business Processes into Analysis Class and Use Case Models: An MDA Approach. Information and Software Technology, 52(9):945-971. https://doi.org/10.1016/j.infsof.2010.03.015

[12] Kriouile, A., Addamssiri, N., & Gadi, T. (2015). An MDA Method for Automatic Transformation of Models from CIM to PIM. American Journal of Software Engineering and Applications, 4(1):1-14. https://doi.org/10.11648/j.ajsea.20150401.11
[13] Object Management Group (OMG), MOF 2.0 QVT. http://www.omg.org/spec/MOF/2.0/.
[14] Luciano, L., Grant, M., & Bran, S. (2003). UML for Real, Design of Embedded Real-Time Systems. Springer Science + Business Media, Inc.
[15] OMG, UML 2.4.1 Superstructure Specification, 2011.
[16] Rosenberg, D., & Stephens, M. (2007). Use Case Driven Object Modeling with UML. APress, Berkeley, USA.
[17] Arlow, J., & Neustadt, I. (2005). UML 2 and the Unified Process. Practical Object-Oriented Analysis and Design, 2nd edn., Addison-Wesley, Reading.

## 8 Authors

**Oualid Betari** is a PhD student from Mohammed First University (Oujda, Morocco). His research interests at the MATSI Laboratory (Applied Mathematics, Signal Processing and Computer Science) include model driven engineering, conceptual design of data warehouses, modeling PHP Frameworks, and modeling Leadership tests. (e-mail: beta.oualid@gmail.com).

**Saida Filali** teaches the concepts of Management at Mohammed First University. Her activities of research in the ERDILI (Research Team in Law and Information Technology and Freedom of Information) focuses on Quality and Marketing. (e-mail: sfilali6@gmail.com).

**Amine Azzaoui** is a PhD student from Mohammed First University. His research interests at the MATSI Laboratory include Business Process Management (BPM) and MDA models for the end-to-end design of strategic and operational dashboards. (e-mail: a.azzaoui@enim.ac).

**Mohamed Amine Boubnad** is a PhD student at Mohammed First University, Laboratory: Economics and Management of Organizations. His research interests include management control systems, interactions between management and information systems. (e-mail: ma.boubnad@gmail.com).

Article submitted 03 July 2018. Final acceptance 15 July 2018. Final version published as submitted by the authors.