International Journal of Computer Trends and Technology
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V70I3P104

Volume 70 Issue 3, 19-26, March 2022

Review Article

A Review Paper on Application of Model-Driven Architecture in Use-Case Driven Pervasive Software Development

Julia N. Korongo[1], Samuel T. Mbugua[2], Samuel M. Mbuguah[3]

[1,2,3]*Department of Information Technology, Kibabii University, Bungoma, Kenya.*

Received Date: 12 February 2022
Revised Date: 01 April 2022
Accepted Date: 05 April 2022

Abstract *- This article explores the Model-Driven Architecture (MDA) approach concerning software modelling during systems development. The evolution of MDA is changing software development into a simpler process with less turnaround time, faster deliverability and greater innovation. MDA facilitates building pervasive software systems from high-level models to descriptions of processes known as Use Cases or Scenarios. The first section addresses the role and the importance of software systems architecture in building robust software systems. The second section describes the concept of MDA, in particular, modelling and how to apply Use Case Scenarios using Unified Modeling Language (UML) during software development. Finally, the paper explains the advantages and disadvantages of the MDA and further observes the challenges of MDA in representing the progression and transformation of information in pervasive software development.*

Keywords *- Model-Driven Architecture, Pervasive Software Development, Software Engineering, Software Modeling, Software Systems Architecture, Use-Case Scenarios.*

I. INTRODUCTION

In a rapidly evolving world driven by digital innovations, pervasive software systems play a critical role in supporting personal devices, home appliances and business applications. While designing and developing better software systems interfaces, the architecture and infrastructure of software systems should be optimised to meet both the user needs and business requirements [1]. Hence, there is a need for software architecture that makes usability and interoperability central to other supporting infrastructures. Studies show that one of the approaches to optimise the development of software systems is to apply Model-Driven Architecture (MDA) during software engineering [2], [3], [4], [5]. On the other hand, references [6], [7] allude that the evolution of MDA is changing software development into a simpler process with less turnaround time, faster deliverability and greater innovation. Therefore, as the pace of technology continues to quicken, as observed by [1], the demands of integrating enterprise-wide existing legacy systems with industry 4.0 technologies and other digital platforms such as social media and e-business systems are thus real.

The demand for pervasive software systems has increased, leading software engineers to devise and improve software architectures and different techniques in the systems development process [1], [5]. According to [8] and [9], end-users are increasingly demanding more from software products than ever before regarding more features, faster runtimes and fewer software errors. Software Engineers have devised different ways by employing new toolsets in developing and delivering the best quality software products. This is because software development is a complex, elaborate and iterative process of understanding, analysis, discovery, and design of systems that requires techniques to increase understanding of the defined set of stakeholders and address the systems requirements. This means that the more complexity developers attempt to address in terms of requirements, the more change that occurs and the more change that occurs, the more complexity in terms of the software pervasiveness [10], [11]. Hence, the development must involve discovering user and business requirements and developing solutions through collaboration by software engineering teams with their end-users [12], [13].

II. METHODOLOGY

This study adopted a qualitative approach. In particular, desk research was conducted to review the literature that provided baseline information about the MDA approach. The study further sought to explore the MDA framework in relation to Use-Case was driven by software systems

development and adopts a descriptive strategy to review literature from different sources, including journal articles, textbooks, conference papers, scientific, social sites and websites. Data presentation is in the form of analytical and textual descriptions using tables, diagrams and content analysis.

III. RESULTS AND DISCUSSION

A. The Role of Software Systems Architecture

Software systems architecture is the foundation that sets up the base for designing and implementing software systems [4], [14]. Software architecture deals with software systems' abstraction, decomposition, composition, style, and esthetics, as observed by [4], [15]. Reference [15] further notes that a software architecture description can be organised around four views and then illustrated by a few selected Use Cases considered in the fifth view, as shown in Fig. 1. This also involves a set of high-level decisions that determine the structure of the software solution (parts of software-to-be and their relationships).

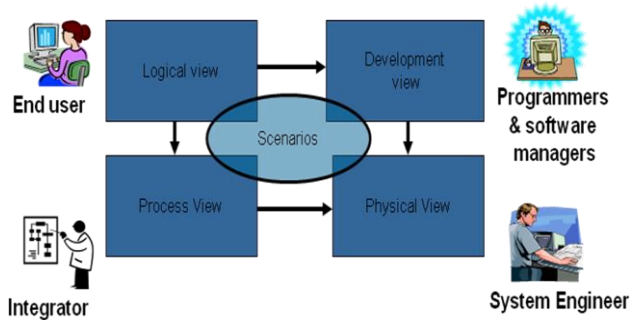

Fig. 1 The 5-Views Architectures

Source: [15]

The model by [15], as shown in Fig. 1, implies that there should be principal decisions made throughout the development and evolution of a software system, and decisions about the design philosophy are made early and affect large parts of the system hard to modify later. Throughout the process of the Software Development Life Cycle (SDLC), many kinds of information are captured, analysed, refined and communicated. There are three types of approaches to software development: SDLC, Agile Approach and Object-Oriented Systems Analysis and Design. These involve different models such as Iterative, Waterfall, Incremental, Rapid Application Development, and Agile methods, among others, for applying these activities. A study by [16] opines that the system modelling approach is key while developing consistent pervasive systems such as those used in the manufacturing industry to facilitate access to the Web. Reference [16] further describes the decomposition of a system using Objective Oriented Project Planning (OOPP) in three dimensions of analysis: the problem, the objective and the activity planning. Thus, [10] further argues that it is important to use well-known solutions that are proven to work for similar problems.

B. The Concept of Model-Driven Architecture

Model-Driven Architecture is a software design approach launched by the Object Management Group (OMG) in 2001 [17]. It is an open and vendor-neutral architectural framework that leverages associated OMG standards (and models specifically) within the SDLC across various domains and technologies. The approach provides a set of guidelines for structuring specifications, which are expressed as models and are said to have partially evolved from the Use Case Scenarios, as shown in Fig. 1. The aim is to model first to enable developers and other stakeholders like end-users and customers to have a complete picture of what software is required. This entails having a complete view of the Use Cases of the business in terms of the processes and the interactions involved while documenting the entire software project. According to [17], the goal of using the MDA framework is guided by the following four questions:

i) What does the software engineering project team currently have in architecture modelling?
ii) What does the team want to know about business goals and initiatives?
iii) What roadmaps or frameworks are necessary to achieve the software engineering project?
iv) How does the team communicate the progress of the software engineering project?

The MDA method leverages the modular architecture throughout the software development process and is divided into three abstract layers of modular transformation as described by [7], [18]. These abstract layers are also represented by [19] as an MDA framework that includes; Computation-Independent Model (CIM), Platform-Independent Model (PIM), and Platform-Specified Model (PSM). Table 1 describes each of the models.

Also called Scenarios or Viewpoints, as indicated in Fig. 1, the three models shown in Table 1 may also be supported by UML diagrams using different modelling techniques and tools [20] in software development. The UML techniques help to enable and facilitate the process of software development. References [3], [21] argue that the term 'architecture' in MDA does not refer to the architecture of the system being modelled but rather to the architecture of the various standards and model forms that serve as the technology basis for MDA. The importance of MDA is the notion of model transformation, which focuses on forwarding engineering, that is, producing code from abstract, human elaborated modelling diagrams like class diagrams and use case diagrams, among others. OMG has defined a specific language for model transformation called Query View Transformation (QVT) language [18] that facilitates the process of forwarding engineering in modelling software systems.

Furthermore, MDA also specifies formal transformations between the PIM and the PSM [7]. This is achieved by adding marks to the PIM to indicate how certain elements can be mapped onto different platforms. Many tools produced by different vendors can exchange UML diagrams, and the PIM-PSM transformations are becoming fully automated [22], thus facilitating efficient software development. Fig. 2 shows how the different components represent the problem domain and the platform involved in the final system method.

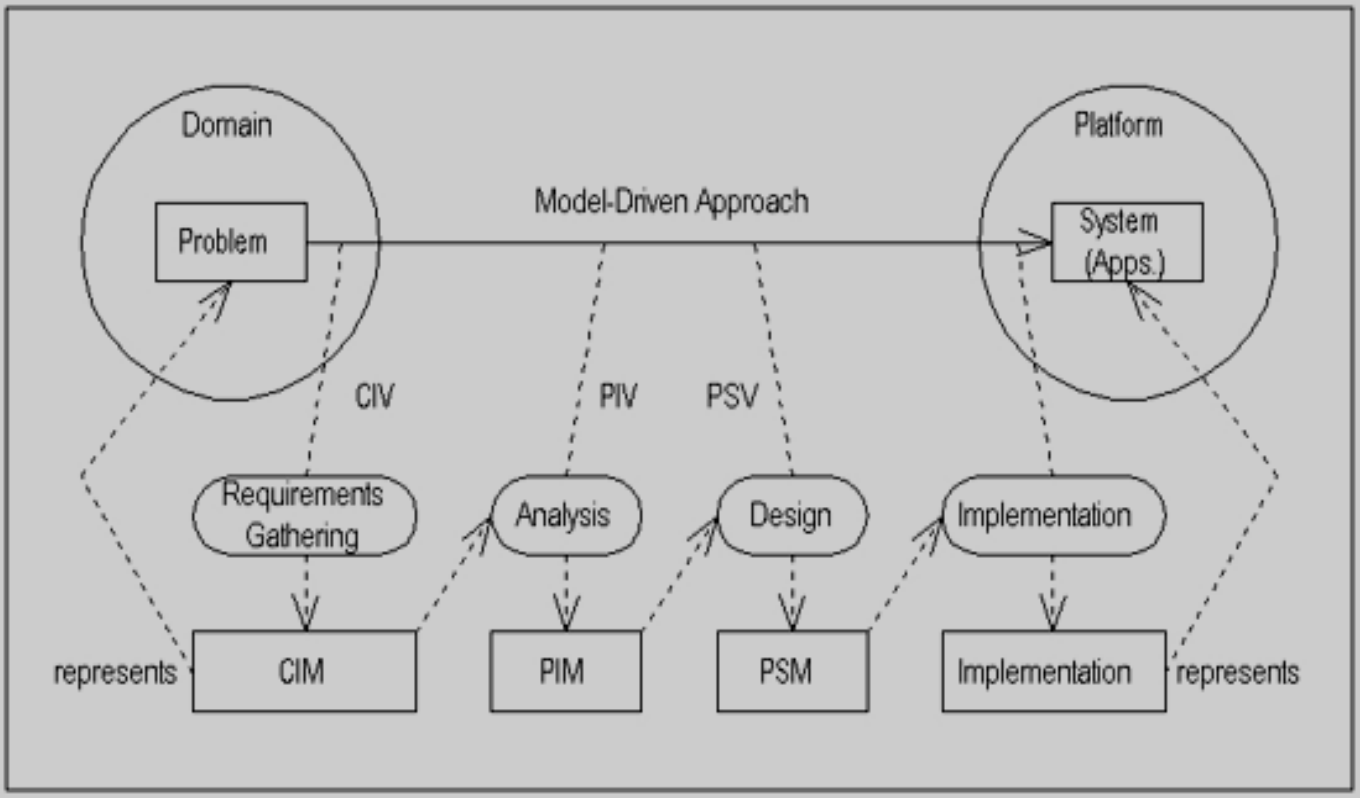

Fig. 2 Foundational Components of the MDA Adapted from: [22]

The components involved in MDA, as shown in Fig. 2, include:

C. The Model

There are three views; Computational Independent view (CIV), Platform Independent View (PIV) and Platform Specific View (PSV), that translate into three models used to build the large-scale solutions named (1) Computational Independent Model (CIM), in which business model does not depend on the system model and implementation details are not present in it; (2) Platform Independent Model (PIM), the name indicates that it is independent of any platform or operating system detail; and (3) Platform Specific Model (PSM), the PIM can be transformed in multiple PSM which is dependent on a particular platform or operating system. Lastly, the Implementation Specific Model (ISM) specifies all implementation details.

D. Model Transformation

As shown in Fig. 2, the different models can be transformed through some sequential steps to develop the system. This begins with requirements gathering transformation to CIM, systems analysis to PIM, systems design to PSM, and systems implementation transforms to ISM. Models are written in a well-defined language that comprises another model used to integrate and transform the model called meta models.

Table 1. Comparison of the three MDA Models

Model Type	How and Where Applicable	Example
Computation-Independent Model (CIM)	• Reflects system and software knowledge from the business perspective. • Models the system in terms of how it will interact with its environment corresponds to the conceptualisation perspective. • Combines requirements and domain models that interact with its environment. • Represents descriptions of functionalities while hiding the technical specifications.	• A specification for business rules, business processes, data vocabulary and requirements for the software. • For modelling specific problems.
Platform-Independent Model (PIM)	• A model of a system that does not have any technology-specific implementation information. • No change from one platform to another. • Design model that describes the internal structure of models without regard to the hosting platform. A common application concept can be extracted independently from the platform target through PIM. • Also allows mapping to one or more platforms.	• A generic description of a software system independent of the hosting platforms.
Platform-specific Model (PSM)	• A model of a system that has technology-specific implementation information. • Changes from one platform to another. • An implementation model adds concepts from the hosting platform to the specific hosting platform. • Combines the PIM model with the specifications of a particular platform. • Allows model to conform to a specific platform and helps generate appropriate source code.	• A description of the system using, e.g. Java or Microsoft .NET technology. • Oracle SQL Developer Data Modeler 21.4.2 and Oracle APEX.
• Transformation techniques (QVT) convert PIMs that specify the operations of software systems to produce PSMs that specify how software systems use the capabilities of the platforms to provide their operations. [20]		

E. The Process of Mapping and Transformation

Mapping is a specification (or transformation of specification) that includes rules and other information for transforming a PIM to produce a PSM for a specific platform. It transforms the models from one layer to another by transforming from one abstracted layer or model to another, then transforming from one model view to a lesser view. The target meta-model can be tapped by adding information to the models. The model type mapping specifies mapping based on the types of model elements. The model instance mapping specifies how specific model elements are to be transformed in a particular manner using marks. At the same time, transformation is a process of converting a PIM, combined with other information, to produce a PSM. This means that a transformation for a model type mapping is a process of converting a PIM to produce a PSM by following the mapping. At the same time, a transformation for a model instance mapping is a process of converting a marked PIM to produce a PSM by following the mapping. Figure 3 shows the transformation phases in pervasive software systems development.

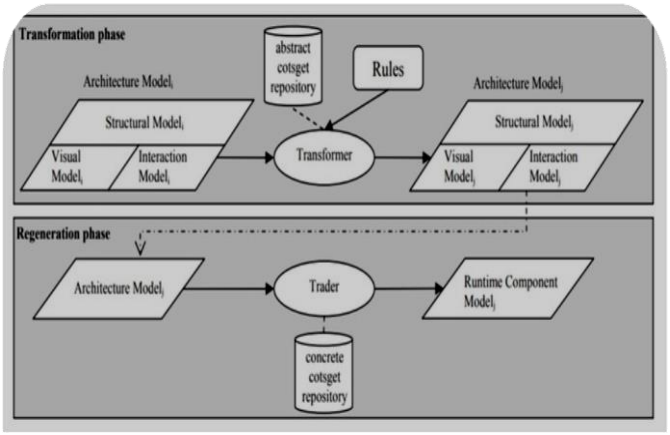

Fig. 3 Transformation of Models in MDA Source [22]

According to [23], two types of transformation approaches can be used in the MDA approach for designing an application, Model to Model (M2M) and Model to Text (M2T), as discussed hereunder:

a) M2M

M2M transforms models from CIM to PIM or PIM to PSM. It is an automatic generation of *t* target models from *s* source models. Each of these models conforms to a reference model, which can be the same for several models. Transformation of M2Mallows having productive modular transformation of models independent of any software execution platform.

b) M2T

M2T is used to model the management process for generating text from different models. It allows the generation of codes from the entry model PSM to a programming language related to the chosen software development platform. The generation of the text is not limited to codes. It varies from documents to manuals as it is independent of the target programming language. In M2T transformations, the text is a form of arbitrary structure and does not conform to any meta-model.

The development of the M2M and M2T meta-models, which initiate the modelling process, is specifically done to reduce the complexity for software engineers, as observed by [21]. These two models then translate the vision and need of the software end-user concerning Use Case Scenarios in the form of actions and interactions, which are then transformed into output software models, a concept referred to as forward engineering. In this case, the descriptions of the user requirements are represented from the system from the graphical point of view by using UMLs.

F. The Concept of Use Case Modeling

In software engineering, a model simplifies reality that forms a description of the real problem or an abstraction that represents and communicates what is important, devoid of unnecessary system details. The fact Because software systems are ubiquitous and key in the current day to day operations [24] and because we cannot comprehend big systems in their entirety, then modelling the real problem helps software engineers to deal with the complexity of the solution being developed by visualising and creating the final software product. There are two types of data modelling:

- Strategic data modelling: this is part of the creation of an information systems strategy that defines an overall vision and architecture for the system; and
- Systems analysis data modelling: using logical data models in systems development.

While modelling software systems, it is important to describe the exchange of information among the key components of the pervasive systems. Reference [16] outlines these specific objectives of a flexible information system as; management and security of information systems, circulation of information, use of appropriate media channels, archiving of information, analysis and effective information processing, and characterisation of the information therein

To achieve the target goals, from requirements analysis to implementation of software systems, software engineers use UML, a non-proprietary modelling language, to design and implement the different data models. Throughout the SDLC process, software engineers use different data models to describe and conceptualise business entities, their processes, and relationships. Further, the three phases of problem-solving emanating from a problem domain to realising a solution include conceptualisation, specification and realisation. According to [25], different modelling techniques are used for each of these phases to represent the various data represented.

First, conceptual data modelling is based on the user and systems requirements [25] or user stories (that produce a requirements model) mainly in the context of an activity model. This model consists of entity types, attributes, relationships, integrity rules, and the definitions of those objects concerning the business needs of the software system. The product of the requirements model is then used as the start point for the specification of the user interface and database design modelling (that produces the analysis model and design model) for the system.

Finally, the product of the design model is used to realise the final product (implementation model) that represents a solution for the software system under development.

G. Application of Unified Modeling Language

UML has a direct relationship with object-oriented analysis and design and support the implementation of MDA [25], [26], [27], [28]. There are four main categories of UML illustrations with several types of diagrams, as presented in Table 2.

Table 2. UML Diagrams and Value to Business

Dimension/ View	Category/ where applicable	Value to Business
Process view	• Use Case and Activity Diagrams • Requirements Analysis	• The flexibility of showing Use Case Scenarios • Understanding and communication
Logical view	• Class Diagrams • Designing Software	• Integration of different Use Case Scenarios in the form of classes and objects • Inheritance and reusability
Dynamic view	• Behavioural Diagrams • Software Implementation	• Interoperability of different Use Cases • Testing, integration and simulation
Deployment view	• Deployment Diagrams • Software Implementation and deployment	• Portability and compatibility of Use Case Scenarios with hardware, network components
Use Case view	• All categories of diagrams are applicable	• The content drives the development of other views using different Use Case Scenarios.

UML is a collection of graphical notations that object methods use to express the designs of systems. It is mainly used for visualising, specifying, constructing and documenting the artefacts in the development of software systems. Software developers apply UML in all SDLC phases while developing pervasive software systems of several different kinds. This includes primarily intensive software systems, database systems and business processes. UML tools have evolved to generate program language code, referred to as forwarding engineering from UML diagrams to support Use Cases and implement specific applications. For each dimension, as shown in Table 2, there are a number of diagrams that denote a view of the software system's model (4+1 view as provided by Kruchen), and these can be described as Process view, Deployment view, Logical view, Dynamic view + use Case view as shown in Fig. 1. The diagram shows the behaviour of a system from different perspectives and how businesses can derive value from using UML.

A demonstration of implementing the descriptions in Table 2 is drawn from the application of Oracle SQL Developer Data Modeler 21.4.2 and Oracle APEX [29], [30], where data modelling occurs at three levels:

a) Physical data model

Physical data model is a physical model or a schema or framework for how data is physically stored in a database (in the form of SQL codes generated using Oracle APEX).

b) Logical data model

Sits between the physical and conceptual levels and allows for the logical representation of data to be separate from its physical storage (this can be expressed using Oracle SQL schemas).

c) Conceptual data model

This model identifies the high-level user view of data (this can be expressed using a Class Diagram using a Data Modeler).

From the application examples, when we map a PIM to a particular platform regarding the Krutchen's 4+1 views, we produce artefacts native to that platform (diagrams, codes, schemas, deployment descriptors, use cases) but also a platform-specific UML model [31]. We do this because the UML model can express the semantics of the platform-specific solution as described later using the examples from Oracle platforms. The Model-Driven method will focus on the graphical part by initiating a simplified design model. It transforms Use Cases into graphical and visual components to help developers construct an application visually. It exploits the abstraction model of the software and later converts the same into a working software or application.

Another example is where software developers create web application architectures (for example, a Website) using the Model-View-Controller (MVC) pattern, where applications are classified into four groups, namely:

1) Portals and Web Applications

MDA can easily build different types of web applications for both front-end and back-end systems.

2) Mobile Devices Applications

Software developers can build and customise front-end modules for smartphones and tablets.

3) Back Office Applications

This allows software engineers to improve internal enterprise operations by building the back-end applications that use back-office modules only that provide functionality for administration and internal use of business data.

4) Graphical User Interfaces (GUIs)

Graphic user interface that facilitates the first point of contact between the user and the system can leverage such a method to induce a visual approach to designing and achieving better application UIs.

H. Advantages of MDA in Pervasive Software Systems Development

a) Flexibility

MDA can derive or generate code from a stable model as the underlying infrastructure shifts over time hence, reduce the need for hand programming.

b) Return on Investment

The reuse of requirements, inheritance of objects and classes, and application and domain models across the software lifespan enhances return on investment.

c) Increased Productivity

in the design and development phases of software projects, the PIM-PSM conceptual separation can potentially provide a better division of tasks between domain and platform experts, facilitating the principle of Separation of Concerns in software engineering.

I. Disadvantages of MDA in Pervasive Software Systems Development

a) Transformation of PIM

PIM is not transformed into PSM in some cases, but it is instead used as a base for generating the source code.

b) Abstraction of PSM

The level of abstraction provided by the PIM is abandoned in favour of PSM in some cases that are then directly interpreted by a compiler that translates them into code.

c) Dimension of Instability

The relatively stable PIM protects software developers when shifting enterprise boundaries. The software developer’s challenge is how to preserve the development investment made in new components when an enterprise boundary shift and the underlying technology needs to change [10], [12], [32].

J. Challenges of MDA

As pointed out by [4], software abstraction has shown to be particularly effective for reasoning about the software's structure, constituent elements and the relationships among them. There is no final data model for a business or user application because of dynamic and heterogeneous business environments that should also match the range of timing to currently used powerful ubiquitous computing elements [11] to handle various pervasive software systems attributes. For instance, a study by [24] indicates that internal software attributes such as cohesion, coupling, and complexities can predict attack ability metrics related to external software attributes in the architectural design software system. Hence, a data model should be considered a living document subject to change in response to the change in business. It is also important to note that the data models will require to be edited, expanded, and retrieved over time; thus, a repository for such models is vital for reuse. This implies that one can incorporate MDA with the Repository Software Systems Architecture when developing pervasive software systems. Data models can be centralised and accessed frequently by other components to access or modify data and the knowledge base to avoid redundancy and model inconsistencies.

IV. CONCLUSION

The development process of pervasive software systems is a complex task that can be made more efficient by using a simplified and flexible approach like MDA. The MDA broadly supports different types of application domains and technology platforms. The idea behind the MDA process is to begin designing the pervasive software systems by assimilating them to a non-technical model that is instead focused on representing its domain structure and functional behaviour, namely, PIM. This is followed by transformation to more technical models, specific for the implementation of the system modelled in different platforms, namely, PSM. Currently, different UML tools support forward engineering from different Use Case Scenarios. However, the challenge of MDA arises due to the dynamic and heterogeneous nature of the requirements in the business environments that should also match the range of pervasiveness in modern computing.

Through UML's modelling technique, MDA has empowered developers to address change and complexity and leverage change and complexity for a competitive advantage by capturing knowledge encoded in models. This approach leads to long-term pervasive software systems architectures that support different metrics such as flexibility of implementations, integration, maintenance, testing, attack ability and simulation, and portability, interoperability and reusability. The observation from this paper is that software abstraction in the form of data models represents information areas of interest referred to as Scenarios or Viewpoints and is, therefore, progressive and transformative. Hence, there is a need to improve methods and techniques or devise new methods and techniques to meet the new pervasive software engineering challenges. The software architecture development process is thus important to deal with both the internal and external attributes of pervasive software systems.

ACKNOWLEDGMENT

The authors would like to thank the following reviewers whose comments have improved this paper:

REFERENCES

[1] M.V. Steen, and A.S. Tanenbaum, Distributed Systems, 3rd ed. CreateSpace Independent Publishing Platform, (2017).

[2] DS. Frankel, Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons, (2015).

[3] DS. Frankel, and J. Parodi (eds), The MDA Journal: Model Driven Architecture Straight From The Masters. Meghan Kiffer Pr, (2015).

[4] E. Kouroshfar, M. Mirakhorli, H. Bagheri, L. Xiao, S. Malek, and Y. Cai, A Study on the Role of Software Architecture in the Evolution and Quality of Software. Conference: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories (MSR), (2015). Available: https://www.researchgate.net/publication/308735174

[5] A. Soylu, and P. De Causmaecker, Merging Model-Driven and Ontology-Driven System Development Approaches Pervasive Computing Perspective, in Proc 24th Intl Symposium on Computer and Information Sciences. (2009) 730–735.

[6] A. Noureen, A. Amjad and F. Azam, Model Driven Architecture Issues, Challenges and Future Directions, Journal of Software, 11 (9) (2016) 924-933. Available: https://doi.10.17706/jsw.11.9.924-933

[7] M. de Miguel, J. Jourdan, S. Salicki, Practical Experiences in MDA Application. In: J.M. Jézéquel, H. Hussmann, S. Cook, (eds) ≪UML≫ 2002 — The Unified Modeling Language. UML 2002. Lecture Notes in Computer Science, vol. 2460. Springer, Berlin, Heidelberg. [Online] Available: https://doi.org/10.1007/3-540-45800-X_11

[8] I. Englander, The Architecture of Computer Hardware, Systems Software and Networking: An Information Technology Approach, 5th ed. Wiley. (2014).

[9] G.D. Vicente, M.C.l. Juan, B.C.P. García-Bustelo and, O.S. Martínez, Progressions and Innovations in Model-Driven Software Engineering. Eds. Portland: Books New Inc. (2013).

[10] R.S. Sangwan, Software and Systems Architecture in Action. Auerbach Publications. (2014).

[11] F. Oquendo, Software Architecture Challenges and Emerging Research in Software-Intensive Systems-of-Systems, in Tekinerdogan B., Zdun U., Babar A. (eds) Software Architecture. ECSA. Lecture Notes in Computer Science, vol. 9839. Springer, Cham. (2016).

[12] N. Rozanski and E. Woods, Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, 2nd ed. Addison-Wesley Professional. (2011).

[13] L. Bass, P. Clements and, R. Kazman, Software Architecture in Practice (SEI n Series in Software Engineering), 3rd ed. Addison-Wesley Professional. (2012).

[14] K. Smolander, Software Architecture Design in Information Systems Development: A Method Engineering View, (2009). [Online]. Available:https://www.researchgate.net/publication/2401790_ Software_Architecture_Design_in_Information_Systems_Development_A_Method_Engineering_View

[15] P. Kruchten, Architectural Blueprints – The "4+1" View Model of Software Architecture. IEEE Software, 12(6) (1995) 42-50.

[16] M. F. Karoui and M. N. Lakhoua, Information Organization of a Flexible Manufacturing based on System Modeling Approach. SSRG - International Journal of Mobile Computing and Application (IJMCA). 7(1) (2020) 1-5. Available: https://doi.org/10.14445/23939141/IJMCAV7I1P101

[17] R. Soley and the OMG Staff Strategy Group, Model Driven Architecture. Object Management Group White Paper, (2001). [Online]. Available: https://www.omg.org/mda/mda_files/model_driven _architecture.htm

[18] Q. Betari, S. Filali, A. Azzaoui and, M.A. Boubnad, Applying a Model-Driven Architecture Approach: Transforming CIM to PIM Using UML. International Journal of Online and Biomedical Engineering (iJOE), 14(9) (2018) 170-181. Available: https://onlinejour.journals.publicknowledgeproject.org/index.php/i-joe/article/view/9137

[19] A. Kleppe, MDA Explained, The Model Driven Architecture: Practice and Promise. Addison-Wesley, (2003).

[20] J.M. Vara, B. Vela, V.A. Bollati and, E. Marcos, Supporting Model-Driven Development of Object-Relational Database Schemas: A Case Study. In: Paige R.F. (eds) Theory and Practice of Model Transformations. ICMT. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 5563 (2009). [Online]. Available: https://doi.org/10.1007/978-3-642-02408-5_13

[21] C. Raistrick, Model Driven Architecture with Executable UML. Cambridge University Press,(2004).

[22] Y. Rhazali, A. El Hachimi, I. Chana, M. Lahmer and, A. Rhattoy, Automate Model Transformation From CIM to PIM up to PSM in Model-Driven Architecture. In Gupta, B. B. (Ed.), Modern Principles, Practices, and Algorithms for Cloud Security, (2020) 262-283.. IGI Global. Available: http://doi:10.4018/978-1-7998-1082-7.ch013

[23] A.W. Brown and J. Conallen, An Introduction to Model Driven Architecture (MDA) Part II: Lessons from the design and use of an MDA toolkit level, (2005). [Online] Available: https://www.ibm.com/developerworks/rational/library/content/RationalEdge/apr05/brown

[24] S. Mbuguah and F. Wabwoba, Attack ability Metrics Model for Secure Service-Oriented Architecture. Lambert Academic Publishing. (2014).

[25] A. D. Shinde, Use Case Modeling for requirement specifications, Seventh Sense Research Group (SSRG) - International Journal of Computer Trends and Technology (IJCTT), 54(1) (2017) 30-34. Available: https://ijcttjournal.org/archives/ijctt-v54p107

[26] (2013) J. Cabot, Modeling Languages. [Online]. Available: https://modeling-languages.com/anybody-using-both-mda-platform-independent-and-platform-specific-models/

[27] M. Brambilla, J. Cabot and M. Wimmer, Model-Driven Software Engineering in Practice, foreword by Richard Soley (OMG Chairman) in Synthesis Lectures on Software Engineering #1. Morgan & Claypool, USA , (2012). Available: http://www.mdse-book.com

[28] C. Francis, P. Wright, J. Carter and I. Wilkie, Model Driven Architecture with Executable UML, 13–9 [4]. (2004).

[29] (2022) The Oracle APEX Website. [Online] Available: https://oracle.github.io/learning-library/developer-library/apex/low-code-development/

[30] The Oracle Academy Website, (2022). [Online]. Available: https://www.oracle.com/tools/downloads/sql-data-modeler-downloads.html

[31] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord and, J. Stafford, Documenting Software Architectures: Views and Beyond, 2nd ed. Addison-Wesley Professional, (2010).

[32] R. Ramanathan and R.K. Kirtana, Handbook of Research on Architectural Trends in Service-Driven Computing, IGI Global, (2014).