Computer Science and Information Technologies
Vol. 3, No. 3, November 2022, pp. 157~168
ISSN: 2722-3221, DOI: 10.11591/csit.v3i3.p157-168

Automatic model transformation on multi-platform system development with model driven architecture approach

Aila Gema Safitri, Firas Atqiya
Department of Informatics, Faculty of Science and Technology, Universitas Muhammadiyah Bandung, Bandung, Indonesia

Article Info

Article history:

Received May 27, 2022
Revised Nov 4, 2022
Accepted Nov 10, 2022

Keywords:

Automatic model transformation
Model-driven architecture
Rapid application development
System prototyping

ABSTRACT

Several difficulties commonly arise during the software development process. Among them are the lengthy technical process of developing a system, the limited number and technical capabilities of human resources, the possibility of bugs and errors during the testing and implementation phase, dynamic and frequently changing user requirements, and the need for a system that supports multi-platforms. Rapid application development (RAD) is the software development life cycle (SDLC) that emphasizes the production of a prototype in a short amount of time (30-90 days). This study discovered that implementing a model-driven architecture (MDA) approach into the RAD method can accelerate the model design and prototyping stages. The goal is to accelerate the SDLC process. It took roughly five weeks to construct the system by applying all of the RAD stages. This time frame does not include iteration and the cutover procedure. During the prototype test, there were no errors with the create, read, update, and delete (CRUD) procedure. It was demonstrated that automatic transformation in MDA can shorten the RAD phases for designing the model and developing an early prototype, reduce code errors in standard processes like CRUD, and construct a system that supports multi-platform.

Corresponding Author:

Aila Gema Safitri
Department of Informatics, Faculty of Science and Technology, Universitas Muhammadiyah Bandung
Soekarno-Hatta Street 752, Bandung 40614, West Java, Indonesia
Email: ailagema@umbandung.ac.id

1. INTRODUCTION

Software development life cycle (SDLC) models have been developed from a crude linear sequential process model to a more dynamic and inventive approach to satisfy customer and market demands more swiftly during the past few decades [1]. Several issues often arise in the SDLC process: system requirements in the customer domain generally take a long time, how to balance the time with the effort required for system implementation, integration between systems, and coordination between teams with different skills [2]. In addition, the conventional software development approach has several limitations, including the necessity to rewrite function code when a technological update occurs [3]. This is one thing that makes the software development process take so long.

In 1991, James Martin published a book about the rapid application development (RAD) approach [4]. According to Martin, the primary objectives of RAD include high-quality systems, rapid development and delivery, and low costs [4], [5]. RAD entails iterative development and prototype construction [1], [6]. It allows for quick prototyping, retargeting, reuse of existing software, and hardware-specific optimization [7]. The RAD technique was created to overcome the limitations of traditional system development methodologies such as waterfall [7], [8]. Compared to traditional software engineering methodologies, using RAD in the application development cycle can promote a faster process and provide high-quality software

[9]. It allows enterprises to cut software development and maintenance expenses [10]. It is suggested that the RAD method be used with other system development techniques and with the help of the right tools for project management [11].

The four-phases of the RAD are requirements planning, user design, construction, and cutover [4], [5], [11]. The RAD phase focuses on the iterative user design process, which includes prototyping, testing, and refining. According to various publications, RAD prototype modelling method consists of five stages: business modelling, data modelling, process modelling, application development, testing, and turnover [6]. The emphasis of the RAD method is on rapid prototype development with small feature sets in early versions. The initial version of the prototype is tested with users to get feedback that will improve the development of the subsequent prototype. The prototype's next version is developed by reusing the templates, tools, processes, and code that existed in the previous version. Therefore, automation tools are used to support the acceleration of the prototype-building process.

Several literature studies on system development using the RAD method, including:

- Development of an agricultural irrigation system for software [7], BAZNAS Zakat Receipt Information System [12], data bank population [13], and an event management system [14]. Most research on RAD applies the model design phase manually using specific tools, particularly for use case designs, activity diagrams, class diagrams, and entity relationship diagrams (ERDs). Similarly, system developers implement framework-based code during the construction of the prototype phase. In addition, some of these studies lack information regarding the duration of each growth phase. The information about the timeline in each phase can demonstrate whether the system implements the RAD approach successfully.
- Research about developing a website for dutatani leveraging iterative models and prototype procedures [9] has addressed the timeline of each RAD phase. Table 1 shows the duration required for implementing the RAD phases in dutatani website development.

According to Table 1, system development using RAD takes 30 weeks or 150 days. The system prototyping phase required the most time and the most significant number of teams.

- Investigate the efficiency of the RAD stage by switching from manual to automatic requirements-gathering techniques using agents [15]. As a result, using agents in the RAD phase, precisely requirements planning, can improve quality, accelerate software engineering procedures, and produce fast, quality-oriented, and efficient computer systems. However, the time required for each stage of RAD implementation was not explained in detail in this study.
- Another research on the RAD approach for constructing a geographic information system (GIS) [16]. Table 2 shows that the development time of GIS took 12 weeks or 84 days. Each stage in Table 2 is completed using manual techniques.
- Research conducted by Fatima *et al*. [11] has attempted to accelerate the RAD process by shifting the requirements-gathering approach from manual to automatic employing agents such as an entity that can independently observe and respond to its surroundings. This technique has increased efficiency in the RAD phase of the project's early stages and even decreased the mistake rate during the requirements gathering phase.
- Paper on model-view-controller (MVC) framework integration into RAD process model to map defined user requirements into MVC architecture [1]. The goal is to simplify the software product delivery process while maintaining the application's quality of service. However, the paper does not include a timeline for each stage of the RAD.

Table 1. Timeline of dutatani website development [9]

Model	Stages	Duration	Number of Teams
Iterative	Planning, Analysis, Main Design	5 Weeks (May-June)	1
System Prototyping	Design, Specific, Implementation, Prototyping	16 Weeks (June-September)	2
Iterative	Integration	9 Weeks (September-October)	1

Table 2. Development time of GIS of industrial centres [16]

Stages	Duration	Number of Teams
Planning	2 Weeks	1
Analysis	1 Week	1
Design	2 Weeks	1
Implementation	7 Weeks	2

The RAD is one of the practical methods for rapid software development. However, it has significant downsides, such as an enormous risk of system failure, less efficiency in meeting customer satisfaction, and more time spent working on different phases [11]. Based on a literature study of several papers on system development using RAD, some implementation difficulties have been identified:

- System development utilizing the RAD technique frequently concentrates on implementing the existing phases in RAD without introducing innovation to accelerate the process in each phase. In general, the results demonstrate that the system was successfully developed based on the RAD phases without demonstrating the novelty of the approach or technical innovation.
- The model design stage still takes a lengthy time. The team does not utilize tools that can automatically generate models. Several technologies have been used to develop the model, including use case, ERD, class diagrams, interface designs, and other model designs. However, the team must still design and construct models manually using these tools.
- The prototype construction phase is the most time-consuming and requires many teams. Multiple backend and frontend development teams must collaborate to create a prototype, starting with building the system architecture, database, tables, backend, and system interface. However, not all teams can complete tasks on time. For example, team A may complete a task in 16 weeks, but Team B may complete it in 12 weeks; the system integration process only starts when all teams have completed their work [9].

This paper discusses how to speed up the stages of RAD and improve the resource efficiency of the development team by producing design models and prototypes in a short time, less prone to errors, and enabling multi-platform system deployment. To do this, we proposed a model-driven architecture based (MDA) automation approach at every stage of RAD, especially when creating models and constructing system prototypes. The conceptual framework and software development process in MDA depends on the model [2], [17]. The model can represent business processes, hardware, software, the environment, and other domain-specific system components [18]. Model-Driven Development stipulates that a system is established and developed based on a model on an independent platform, which is then converted to a specific platform [17], [19].

The object management group (OMG) developed MDA as a software development approach in 2001 [2], [3], [18], [19]. MDA or model-driven architecture is built on separating business logic and implementation logic [19]. It provides a collection of tools for the process of abstracting the general aspects of an application in order to improve the software development process [3]. The model provides a methodical and comprehensive picture of an application that must be created utilizing software and programming languages [20]. A model simplifies something so that it may be viewed, manipulated, and reasoned about, allowing us to comprehend the inherent complexity of the subject under study [21]. A model can be described in several types of expressions, including unified modelling language (UML) class diagrams, ERDs, and user interface images [18].

Several previous studies indicate that the MDA method can increase efficiency in the software development process [2], [3]; supports interoperability against design-time [3], [21], and supports multi-platform [3], [22]; reduces effort-time and generates error-free code [17]. Amr *et al*. [19] have described the methods and procedures for semi-automatic transformation of the CIM model presented by the BPMN source model (Business Process Model and Notation) into the PIM target model presented by the class diagram on the "COVID-19 patient management" business process. The research by Ahmed *et al*. [22] focuses on MDA issues taking precedence over testing or SPL when automating test scripts. This study uses two case studies to show the PSM taxonomy in the well-established test script domain. It also lists several relevant research challenges that need to be solved in order to solve this problem. Aksakalli *et al*. [23] present a model-driven strategy for the automated deployment of microservices to reduce execution and communication costs. Using a series of transformation rules and recommendations, Habba *et al*. [24] describe an MDA approach to transforming a group of BPMN models into a UML class diagram. By examining a group of models at the source level containing a substantial number of BPMN meta-model elements, MDA can contribute to the alignment process inside an organization [24]. Amdahl *et al*. [25] employ the MDA methodology to design the architecture between the business process and technological layer to support collaborative business processes. It proposes a graphical DSL (BPMN4Coll) for collaborative business processes based on BPMN [25].

The MDA Framework increases the level of abstraction and reusability, allowing applications to become independent of a single platform and seamlessly deploy on multiple technologies [22]. The function of the abstraction level in MDA is software reuse with an emphasis on interoperability at design time [21]. Modelling and abstraction enable us to comprehend and specify a system in great detail, for instance, how a system can be implemented on a specific technology or platform [18]. Figure 1 illustrates a collection of OMG-defined layers and transformations that serve as MDA's conceptual framework and terminology [2]. In the MDA layers, various types of models are transformed. A transformation transforms a model from one

degree of abstraction to another, such as a more abstract view to a less abstract view, by adding detail via transformation rules [26]. The transformation process can be implemented in two ways: model-to-model (the transfer from CIM to PIM or PIM to PSM) and model-to-text (the production of code from PSM to a particular programming language as a target) [26].

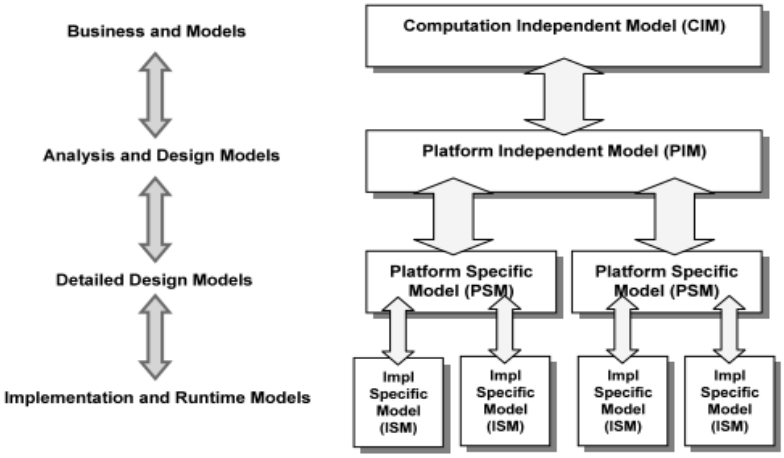

Figure 1. Layers and transformation process of MDA [2]

Computation independent model (CIM) is a model-related or business domain layer. This layer is concerned with people, locations, things, and the fundamental laws of the domain [18]. Use cases are derived from stakeholder requirements [27]. PIM is a sophisticated model that distinguishes logic from technology implementation [17]. PIM is platform and technology-independent, enabling the development of applications compatible with many systems [3]. In PIM, model translation occurs in high-level abstraction models that are turned into low-level abstractions dependent on specific technologies or platforms [2]. Meta models translate CIM models and their interactions into system models at the PIM level [26]. All parameters must be completed for PIM to be considered validated [27].

Platform-specific model (PSM) is generated through a specific mapping procedure or model transformation in PIM [2], [28]. PSM's high-level abstraction model emerged from transforming PIM into a technology-dependent, low-level abstraction model [2]. The automated mapping of meta-models took place at the PIM to PSM layer [22]. PSM is a high-level model with more general semantics, including- create, read, update, and delete (CRUD) actions mapped to native API [22]. It describes the system's technology architecture as illustrated by a particular implementation [17]. Mapping is handy for generating a collection of to-be-coded rules. This code is generated when changing the PSM model into code associated with technology or tools [27], [29].

The following step is to generate code from PSM using a code generator. Code generators automate repetitive coding and configuration operations, fulfil abstract implementation details, and decrease development expenses and time [30]. Code generators can produce entire code or skeletons for programmers to complete [26]. The code generator automatically translates an existing model, including a set of rules, into a programming language. Finally, both artefacts (code and model) can be edited to accommodate model modifications [26]. Generators enable the creation of web applications with complete functionality, a responsive interface that can be used on any device, complete source code that can be scaled and customized, and, most importantly, substantial time and cost savings when developing software projects [30].

2. RESEARCH METHOD

There are three implementation techniques for RAD: iterative development, system prototyping, and throwaway prototype [8]. As depicted in Figure 2, system prototyping is used in this study's RAD modelling technique [8]. The prototyping system development techniques' analysis, design, and implementation phases aim to develop the prototype system rapidly. The initial iteration of the prototype contains simply the bare necessities for the user [9]. Implementing this early prototype for users is intended to elicit feedback and comments. The developer will evaluate the feedback and comments and use them to do analysis, design, and reimplementation for the next version of the prototype [9]. This iteration will continue until the developer,

user, and funding team concur that the system satisfies all of the organization's needs and feature requirements [8]. The system development process is confined to the creation of the initial prototype and implementation for the user. This research does not involve the cutover stage and creating the final version of the prototype.

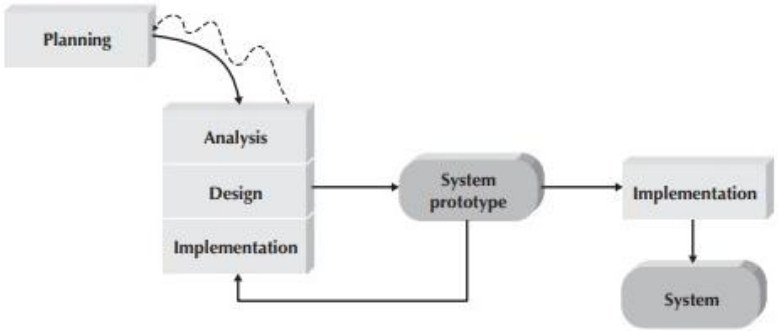

Figure 2. Stages of the RAD model using system prototyping [8]

According to Figure 2, we proposed a method for incorporating the MDA process into the RAD model. The accelerated stages are model design (PIM layer) and prototype construction (PSM layer). Figure 3 illustrates the MDA layers utilized at each RAD level. The model will be produced automatically in the PIM and PSM layers using the JHipster tool. This tool is utilized to rapidly design, develop, and deploy web applications and microservice architectures. Jhipster has a client-server-side technological stack. The client side of this prototype was developed with angular, while the server side was developed with spring, which is comparable to Java programming.

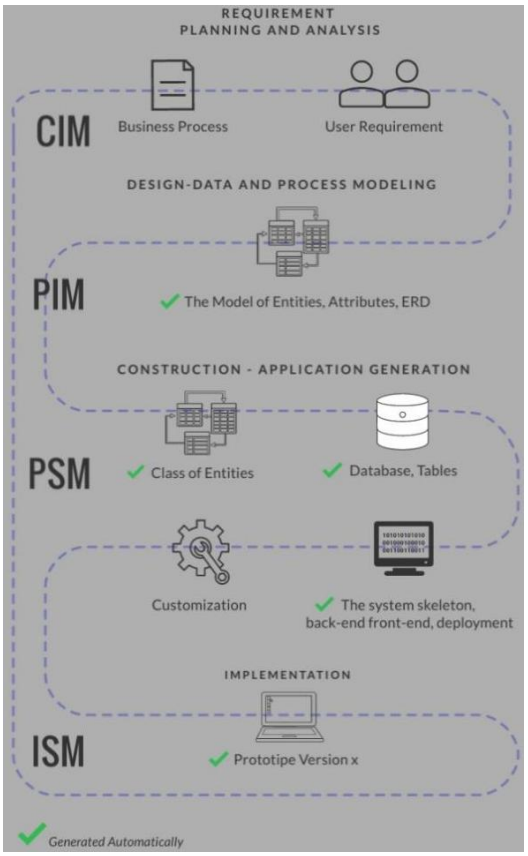

Figure 3. The integration of the MDA process into the RAD prototype model

MDA provides an automatic transformation method based on patterns between abstraction levels [18]. The transformation is performed under a set of rules to provide an output model that conforms to the target meta-model [24]. The pattern was established based on previously proven criteria, such as business and user requirements, which constituted the rules for use cases, entities, and their relationships. MDA performs model transformation as part of a disciplined and effective software development process [2]. The source model corresponding to the source meta-model and the target meta-model are inputs for the model transformation [24]. A well-designed model facilitates the transformation process on a particular platform [17].

3. RESULTS AND DISCUSSION

This section will demonstrate the outcomes of integrating the MDA into the RAD method. The focus of the result was the model transformation process from CIM to PIM and PIM to PSM. A transformation is a mechanism for changing models from one degree of abstraction to another, often from a more abstract perspective to a less abstract one, by adding more detail provided by the transformation rules [26]. Table 3 displays the outcomes of applying the MDA to the RAD model. Planning and analysis are located within the CIM layer. The business process was transformed into an ERD model at the PIM layer. The PSM layer produces an early prototype automatically based on the ERD model. This prototype is compatible with various systems (web and mobile). The final step involved providing the user with the prototype. At this time, we have collected user feedback, which will be used to develop the next version of the prototype.

Table 3. The result of incorporating the MDA into the RAD model using system prototyping

Stages of RAD	Layer of MDA	Automated Process with MDA	Duration	Number of Teams
Planning and Analysis	CIM	-	2 weeks	2
Design	PIM	√	1 day	1
System Prototype	PSM	√	6 days	1
Implementation	ISM	-	2 weeks	1

According to Table 3, all phases of RAD take approximately five weeks to complete. The initial phase, planning and analysis, is approximately two weeks, depending on the length of the discussion with the user and the study of the system's needs. The second stage, the design phase, is a translation from CIM to PIM that yields a design model in the form of an ERD, including sixteen interconnected entities. The ERD is the foundation for transforming the model from PIM to PSM. The third phase is constructing an initial prototype based on the PIM-generated model. A skeleton system was generated for the backend and front end at the PSM layer. Creating a system skeleton as an early prototype takes approximately three to four minutes.

In terms of duration, the customization of the UI and features are the most time-consuming step. The duration of customizing depends on the level of comprehension and technical expertise of the development team regarding the system's skeleton design. The construction of the initial prototype and its customization required six days. The technique of automatically producing prototypes at the RAD stage can save time and reduce system function errors, such as the CRUD procedure for each entity and its relationships. Several research findings on the construction of RAD-based systems indicate that system prototyping is the most time-consuming and team-intensive technique. It is because the development team must manually code the application. At the ISM layer, the final phase concludes with the implementation of the initial prototype for the user. This procedure necessitates user validation of the business logic system agreed upon throughout the design and requirements phase. User feedback gathered throughout the implementation phase will be utilized to evaluate the model and create the next version of the prototype.

3.1. Computation independent model (CIM)

In the CIM layer, we collect requirements from user interviews and conduct several literature searches related to the lecturer activity monitoring system. Admin can manage user registration and master data such as activity types, academic year, faculty, and study program. The lecturer reports the activities by filling out the form activities description and attaching documentation in the form of an image and a file. Tridharma's activities include teaching, research, community service, and other academic activities such as workshops, training and seminars. The lecturer's activities were recorded and documented by the system. A mobile app allows university executives to keep track of lecturer activities.

3.2. Platform independent model (PIM)

The PIM model is constructed using use cases and requirements specified in CIM [27]. PIM provides a system model abstraction level that encapsulates key domain characteristics as classes and entity properties [2]. During this phase, JHipster Domain Language (JDL) Studio is utilized to create entities for the data modelling process. Figure 4 depicts the sample of creating an entity *penelitian* and *pengabdian*, and the relationship between entities.

```
entity Penelitian{
    judulPenelitian String required maxlength(255)
    nilaiPendanaan BigDecimal required
    sumberPendanaan String required maxlength(100)
    mitra String required maxlength(255)
    foto ImageBlob
    linkDokumen TextBlob
}

entity Pengabdian{
    judulPengabdian String required maxlength(255)
    nilaiPendanaan BigDecimal required
    sumberPendanaan String required maxlength(100)
    mitra String required maxlength(255)
    foto ImageBlob
    link TextBlob
}
```

```
relationship OneToOne {
    Dosen{user(login)} to User
}

relationship OneToMany{
   Fakultas to Prodi{fakultas(namaFakultas) required}
   Prodi to Dosen{prodi(namaProdi)}
   Tahun to Penelitian{tahun(tahun)}
   Tahun to Pengabdian{tahun(tahun)}
   TahunAjaran to Pengajaran{tahunAjaran(tahunAjaran)}
   Pengajaran to PengajaranDetil{mataKuliah(namaMatakuliah)}
}
```

Figure 4. Design entities and relationships between entities

Each entity has properties, data type, length, and relationships between entities. After constructing the entities and relationships, we generate the ERD using JDL studio. Figure 5 displays the 16 entities constructed based on the business process's analysis and provisions. The PIM layer produces an ERD design and a JDL file, including entity classes, relations, data types, and field validation code. This JDL file will be utilized in the application development process. The ERD model represents the outcomes of mapping from the CIM layer's abstraction level to entities and their relationships. This ERD model is implementable on numerous platforms.

3.3. Platform specific model (PSM)

The process in PSM explains a high-level abstraction model that is changed into a low-level abstraction based on the technology or tools employed. The model is converted into a specific platform of technology in the form of program code. Code generators were utilized to generate an implementation from the model, allowing for proper model-driven software development. Code generators, similar to compilers, support a particular source language, translate it into a target language, and are written in a particular language [26]. In this stage, the model transformations in the PSM layer were carried out automatically with the help of the code generator Jhipster Domain Language (JDL) - Studio. JDL is a unique JHipster language domain that uses a user-friendly syntax to define all applications, implementations, entities, and their relationships in one or more software files. The JDL executes a file that contains entity classes, relations, data types, and field validation generated in the PIM layer. The ERD model was transformed into a prototype that can be functionally tested. The model is converted into code, resulting in a system skeleton, a database, tables, frontend, and backend.

To initialize the application and produce the platform based on the ERD model, we run JHIpster. The initial stage determines the application type: monolithic; the application name: sidoktor; the authentication type; the database type: PostgreSQL, the generating backend: Maven; the asynchronous message broker; the API development using OpenAPI-Generator; the generating client framework: angular, and the generating admin UI: typescript. Figure 6 shows the process of creating the system skeleton. This procedure took around four minutes, depending on the laptop or personal computer's specifications.

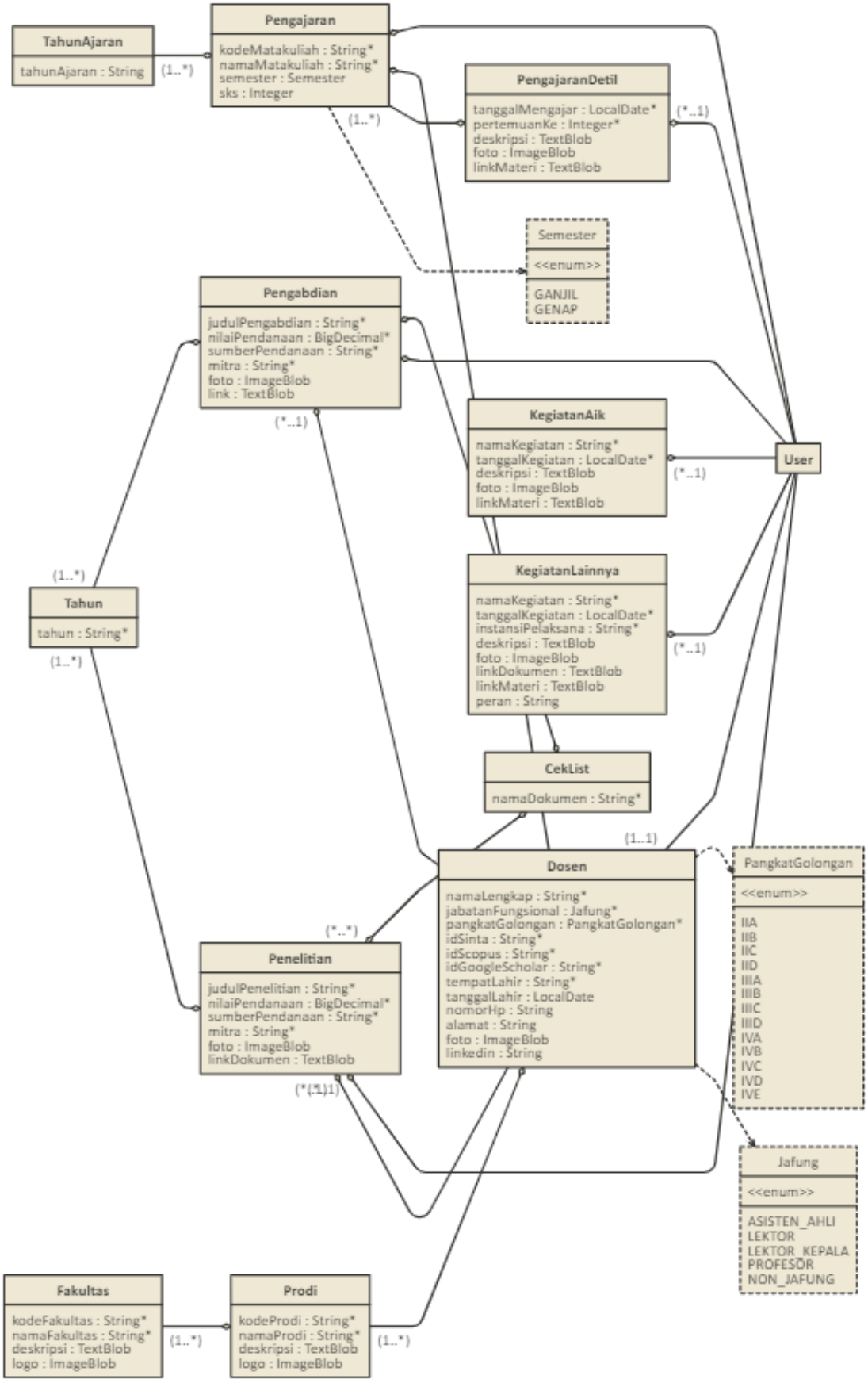

Figure 5. The auto-generated ERD in the PIM layer

```
   create src\main\webapp\i18n\en\sessions.json
   create src\main\webapp\i18n\en\settings.json
   create src\main\webapp\i18n\en\user-management.json
   create src\main\webapp\i18n\in\health.json
   create src\main\webapp\i18n\in\reset.json
   create src\main\webapp\i18n\en\activate.json
   create src\main\webapp\i18n\en\global.json
   create src\main\webapp\i18n\en\health.json
   create src\main\webapp\i18n\en\reset.json
Git repository initialized.

Changes to package.json were detected.
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------< id.ac.umbandung.sidoktor:sidoktornew >----------------
[INFO] Building Sidoktornew 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- frontend-maven-plugin:1.12.0:install-node-and-npm (install-node-and-npm) @ sidoktornew ---
[INFO] Installing node version v14.17.1
[INFO] Copying node binary from C:\Users\Windows\.m2\repository\com\github\eirslett\node\14.17.1\node-
de.exe
[INFO] Installed node locally.
[INFO] Installing npm version 7.18.1
[INFO] Unpacking C:\Users\Windows\.m2\repository\com\github\eirslett\npm\7.18.1\npm-7.18.1.tar.gz into
[INFO] Installed npm locally.
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.079 s
[INFO] Finished at: 2022-01-12T07:26:45+07:00
```

Figure 6. The procedure for automatically generating the system skeleton

The constructed skeleton has produced the admin dashboard and utility menus, such as user management, logging, system login, and API documentation. After completing the skeleton, the JDL files are imported to create the entities, database, frontend, and backend. The process of turning JDL files into entities, tables, APIs, and user interfaces is depicted in Figure 7. This process takes approximately three minutes.

```
Found the .jhipster\KegiatanLainnya.json configuration file, entity can be automatically generated!

     info Creating changelog for entities TahunAjaran,Tahun,Fakultas,Prodi,Dosen,CekList,Penelitian,Pengabdian
    force .yo-rc.json
    force .jhipster\TahunAjaran.json
    force .jhipster\Pengajaran.json
    force .jhipster\Tahun.json
    force .jhipster\PengajaranDetil.json
    force .jhipster\KegiatanAik.json
    force .jhipster\Fakultas.json
    force .jhipster\KegiatanLainnya.json
    force .jhipster\Prodi.json
    force .jhipster\Dosen.json
    force .jhipster\CekList.json
    force .jhipster\Penelitian.json
    force .jhipster\Pengabdian.json
   create src\test\gatling\user-files\simulations\TahunAjaranGatlingTest.scala
   create src\test\gatling\user-files\simulations\TahunGatlingTest.scala
   create src\test\gatling\user-files\simulations\FakultasGatlingTest.scala
   create src\main\webapp\app\entities\tahun-ajaran\tahun-ajaran.model.ts
   create src\main\java\id\ac\umbandung\sidoktor\service\impl\TahunAjaranServiceImpl.java
   create src\main\webapp\app\entities\tahun\service\tahun.service.ts
   create src\main\java\id\ac\umbandung\sidoktor\domain\Tahun.java
   create src\main\webapp\app\entities\tahun-ajaran\delete\tahun-ajaran-delete-dialog.component.ts
   create src\main\webapp\app\entities\tahun-ajaran\list\tahun-ajaran.component.html
   create src\main\java\id\ac\umbandung\sidoktor\service\dto\TahunAjaranDTO.java
   create src\main\java\id\ac\umbandung\sidoktor\web\rest\TahunResource.java
   create src\main\webapp\app\entities\tahun\update\tahun-update.component.html
   create src\main\webapp\app\entities\tahun-ajaran\detail\tahun-ajaran-detail.component.spec.ts
   create src\main\webapp\app\entities\tahun-ajaran\detail\tahun-ajaran-detail.component.html
   create src\main\java\id\ac\umbandung\sidoktor\domain\Fakultas.java
```

Figure 7. The process of generating entities from a JDL file

The entity model and tables produced in the preceding phase are automatically transformed into entity classes on the backend. Figure 8 demonstrates the transformation of a *dosen* entity into a *dosen* class, with multiple properties extracted from the *dosen* table's column. These entity classes are implemented in Java on the backend, whereas TypeScript and Angular are used on the frontend.

```
 * A Dosen.
 */
@Entity
@Table(name = "dosen")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Dosen implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "sequenceGenerator")
    @SequenceGenerator(name = "sequenceGenerator")
    private Long id;

    @NotNull
    @Size(max = 255)
    @Column(name = "nama_lengkap", length = 255, nullable = false)
    private String namaLengkap;

    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(name = "jabatan_fungsional", nullable = false)
    private Jafung jabatanFungsional;

    @NotNull
    @Enumerated(EnumType.STRING)
    @Column(name = "pangkat_golongan", nullable = false)
    private PangkatGolongan pangkatGolongan;
```

Figure 8. *Dosen* entity transformation into *dosen* class

In addition to generating a set of classes and properties for each entity. The PSM layer transformation generates methods that insert, update, and delete system functions. Figure 9 shows an example of several methods produced to add, update, and delete entity of *penelitian*.

```
public Dosen kegiatanPenelitians(Set<Penelitian> penelitians) {
    this.setKegiatanPenelitians(penelitians);
    return this;
}

public Dosen addKegiatanPenelitian(Penelitian penelitian) {
    this.kegiatanPenelitians.add(penelitian);
    penelitian.setKetuaPeneliti(this);
    return this;
}

public Dosen removeKegiatanPenelitian(Penelitian penelitian) {
    this.kegiatanPenelitians.remove(penelitian);
    penelitian.setKetuaPeneliti(null);
    return this;
}

public void setKegiatanPenelitians(Set<Penelitian> penelitians) {
    if (this.kegiatanPenelitians != null) {
        this.kegiatanPenelitians.forEach(i -> i.setKetuaPeneliti(null));
    }
    if (penelitians != null) {
        penelitians.forEach(i -> i.setKetuaPeneliti(this));
    }
    this.kegiatanPenelitians = penelitians;
}
```

Figure 9. The process of generating the methods for an entity of *penelitian*

During the customization phase, the developer can modify the frontend and backend based on the business process requirements and each user's role. Customizing the class of entities, methods, and HTML files permits user interface enhancement. Business processes determine the amount of time necessary for customization. In this study, the process of customizing takes approximately six days.

3.4. Implementation spesific model (ISM)

After the backend and frontend have been customized, the production phase is launched. In just three minutes, the prototype was developed. Four minutes were required to deploy the prototype to Heroku. During the prototype testing, administrators execute numerous scenarios, such as activating users for lecturers and inserting master data such as academic year, semester, faculty name, and study program. Several scenarios for lecturers include user registration, input for tri-dharma activities, and monitoring activities via a mobile application. The preliminary prototype trial period lasted two weeks. The results show that all functional systems can run smoothly and without errors when adding, updating, and deleting the data.

4. CONCLUSION

This study has successfully demonstrated that integrating MDA into the RAD phase can accelerate the model and prototype design process. It took approximately five weeks to design and develop the multi-platform system based on the RAD method. This timeline excludes the phases of iteration and cutover. The automatic transformation of the model can generate the model and skeleton system in minutes. The initial prototype was produced after the customization phase was completed. The amount of time necessary for customization depends on the complexity of the business process and the number of frontend and backend features desired by the user. The customization process is not a serious issue as long as the development team understands the system's skeleton architecture, has technical proficiency, and has Java programming skills, particularly with the Spring framework. The MDA implementation must be based on a mature design model that describes the user's primary requirements. If there are modifications to business processes, it may be necessary to alter the model design. So, the RAD phase will move into an iteration phase, where the model will be redesigned, the prototype will be rebuilt, and the process will be repeated.

ACKNOWLEDGEMENTS

The authors would like to thank the Indonesian Ministry of Education, Culture, Research, and Technology (Kemdikbud) for supporting grants for lecturers through the SIMLITABMAS PDP scheme, which made it possible for us to undertake this research. The researchers from the Information and Communication Technology Research Centre, Bandung Institute of Technology contributed to system development.

REFERENCES

[1] C. Ramos, S. Ganesan, and R. Caytiles, "The integration of MVC framework in rapid application development (RAD) process model," *International Journal of Software Engineering and Its Application*, vol. 12, no. 1, pp. 57–66, 2018, [Online]. Available: http://dx.doi.org/10.21742/ijseia.2018.12.1.05.

[2] A. W. Brown, J. Conallen, and D. Tropeano, "Introduction : Models, modeling, and model-driven architecture (MDA)," in *Model Driven Software Development*, S. Beydeda, M. Book, and V. Gruhn, Eds. Springer, 2005, pp. 1–16.

[3] G. Sebastián, J. A. Gallud, and R. Tesoriero, "Code generation using model driven architecture: A systematic mapping study," *Journal of Computer Languages*, vol. 56, 2020, doi: 10.1016/j.cola.2019.100935.

[4] J. Martin, *Rapid application development*. Macmillan, 1991.

[5] P. Beynon-Davies, C. Came, H. Mackay, and D. Tudhope, "Rapid application development (Rad): An empirical review," *European Journal of Information Systems*, vol. 8, no. 3, pp. 211–232, 1999, doi: 10.1057/palgrave.ejis.3000325.

[6] K. Ali, "A study of software development life cycle process models," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 1, Jan. 2017.

[7] A. K. Nalendra, "Rapid application development (RAD) model method for creating an agricultural irrigation system based on internet of things," *IOP Conference Series: Materials Science and Engineering*, vol. 1098, no. 2, p. 022103, 2021, doi: 10.1088/1757-899x/1098/2/022103.

[8] A. Dennis, B. H. Wixom, and R. M. Roth, *System analysis and design Sixth edition*, 6th ed. United States of America: John Wiley & Sons, Inc, 2014.

[9] R. Delima, H. B. Santosa, and J. Purwadi, "Development of Dutatani website using rapid application development," *IJITEE (International Journal of Information Technology and Electrical Engineering)*, vol. 1, no. 2, 2017, doi: 10.22146/ijitee.28362.

[10] R. Naz and M. N. A. Khan, "Rapid applications development techniques: A critical review," *International Journal of Software Engineering and its Applications*, vol. 9, no. 11, pp. 163–176, 2015, doi: 10.14257/ijseia.2015.9.11.15.

[11] F. Fatima, M. Javed, F. Amjad, and U. G. Khan, "An approach to enhance quality of the RAD model using agents," *The International Journal Of Science & Technoledge*, vol. 2, no. 13, pp. 202–210, 2014.

[12] M. Tabrani, H. Priyandaru, and Suhardi, "Application of the rapid application development method to the Baznas aakat receipt information system in Karawang," *Jurnal Teknologi Dan Open Source*, vol. 4, no. 1, pp. 78–84, 2021, doi: 10.36378/jtos.v4i1.1365.

[13] L. Fitriani, N. E. Berlianti, R. Cahyana, and W. Baswardono, "Information system design of data bank population using rapid

application development," *IOP Conference Series: Materials Science and Engineering*, vol. 1098, no. 3. p. 032049, 2021, doi: 10.1088/1757-899x/1098/3/032049.
[14] B. Julian, A. Triayudi, and Benrahman, "User satisfaction analysis for event management systems using RAD and PIECES framework," *IOP Conference Series: Materials Science and Engineering*, vol. 1088, no. 1, p. 012024, 2021, doi: 10.1088/1757-899x/1088/1/012024.
[15] F. Fatima, M. Javed, F. Amjad, and G. U. Khan, "An approach to enhance quality of the RAD model using agents," *Journal of American Science*, vol. 14, no. 9, pp. 47–55, 2018, [Online]. Available: http://www.jofamericanscience.orgonline.
[16] G. W. Sasmito, D. S. Wibowo, and D. Dairoh, "Implementation of rapid application development method in the development of geographic information systems of industrial centers," *Journal of Information and Communication Convergence Engineering*, vol. 18, no. 3, pp. 194–200, 2020, doi: 10.6109/jicce.2020.18.3.194.
[17] F. Deeba, S. Kun, M. Shaikh, F. A. Dharejo, S. Hayat, and P. Suwansrikham, "Data transformation of UML diagram by using model driven architecture," *2018 3rd IEEE International Conference on Cloud Computing and Big Data Analysis, ICCCBDA 2018*, pp. 300–303, 2018, doi: 10.1109/ICCCBDA.2018.8386531.
[18] L. M. Favre, "Object management group model driven architecture (MDA) MDA guide rev. 2.0," 2014. doi: 10.4018/978-1-61520-649-0.ch002.
[19] M. F. Amr, N. Benmoussa, K. Mansouri, and M. Qbadou, "Transformation of the CIM model into A PIM model according to the MDA approach for application interoperability: Case of the 'COVID-19 patient management' business process," *International journal of online and biomedical engineering*, vol. 17, no. 5, pp. 49–68, 2021, doi: 10.3991/ijoe.v17i05.21419.
[20] M. Stepha and J. R. Cordy, "Model-driven evaluation of software architecture quality using model clone detection," *Proceedings - 2016 IEEE International Conference on Software Quality, Reliability and Security, QRS 2016*, pp. 92–99, 2016, doi: 10.1109/QRS.2016.21.
[21] S. J. Mellor, K. Scott, A. Uhl, and D. Weise, *MDA distilled: Principles of model-driven architecture*. Addison-Wesley Professional, 2004.
[22] A. H. Ahmed, A. A. A. Sidahmed, and R. B. Eltoum, "Automation of test scripts in software product line using model driven architecture," *Proceedings - 2015 International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering, ICCNEEE 2015*, pp. 62–66, 2016, doi: 10.1109/ICCNEEE.2015.7381429.
[23] I. K. Aksakalli, T. Celik, A. B. Can, and B. Tekinerdogan, "A model-driven architecture for automated deployment of microservices," *Applied Sciences (Switzerland)*, vol. 11, no. 20, 2021, doi: 10.3390/app11209617.
[24] M. Habba, M. Fredj, and S. B. Chaouni, "Aligning software system level with business process level through model-driven architecture," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 10, pp. 174–183, 2021, doi: 10.14569/IJACSA.2021.0121020.
[25] L. Amdah, N. Essadi, and A. Anwar, "A model-driven architecture for collaborative business processes," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 8, pp. 719–725, 2021, doi: 10.14569/IJACSA.2021.0120883.
[26] G. Paolone, M. Marinelli, R. Paesani, and P. Di Felice, "Automatic code generation of MVC web applications," *Computers*, vol. 9, no. 3, pp. 1–29, 2020, doi: 10.3390/computers9030056.
[27] M. Menghin, N. Druml, C. Steger, R. Weiss, H. Bock, and J. Haid, "Development framework for model driven architecture to accomplish power-aware embedded systems," *Proceedings - 2014 17th Euromicro Conference on Digital System Design, DSD 2014*, pp. 122–128, 2014, doi: 10.1109/DSD.2014.30.
[28] Pedro De Almeida, "Model driven architecture: improving software development productivity in large-scale enterprise applications," University of Fribourg Switzerland, 2008.
[29] A. Karkouch, H. Mousannif, H. Al Moatassime, and T. Noel, "A model-driven architecture-based data quality management framework for the internet of things," *Proceedings of 2016 International Conference on Cloud Computing Technologies and Applications, CloudTech 2016*, pp. 252–259, 2017, doi: 10.1109/CloudTech.2016.7847707.
[30] Y. P. Atencio, M. J. Ibarra, J. H. Marin, and E. H. Holguin, "Automatic generation of web applications for information systems," *Journal of Physics: Conference Series*, vol. 1860, no. 1, 2021, doi: 10.1088/1742-6596/1860/1/012019.

BIOGRAPHIES OF AUTHORS

Aila Gema Safitri received her master's degree from Bandung Institute of Technology, School of Electrical Engineering and Informatics. She is currently a lecturer at Universitas Muhammadiyah Bandung, Department of Informatics, Faculty of Science and Technology. She also works as a researcher at Information and Communication Technology Research Centre, Bandung Institute of Technology. Her research area covers information systems, software engineering, e-learning systems, multimedia and games. She can be contacted at email: ailagema@umbandung.ac.id.

Firas Atqiya is a Lecturer in the Department of Informatics, Faculty of Science and Technology at Universitas Muhammadiyah Bandung. Her academic credentials are M.Si., M.Sc. Her research area includes artificial intelligence and programming. She holds master's degrees in Computational Science from Institut Teknologi Bandung and Applied Mathematics from Kanazawa University. She can be contacted at email: firasatqiya@umbandung.ac.id.