UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DIPARTIMENTO DI INGEGNERIA E SCIENZE DELL'INFORMAZIONE E MATHEMATICA

Dottorato di Ricerca in Information and Communication Technology

Emerging computing models: algorithms, software architectures and intelligent systems

XXXV ciclo

Titolo della tesi

Cloud-Based Low-Code Model Transformations Composition and Execution

SSD INF/01

Dottorando

Apurvanand Sahay

Coordinatore del corso

Prof. Vittorio Cortellessa

Tutor

Prof. Davide Di Ruscio

Co-Tutor

Prof. Alfonso Pierantonio

a.a. 2021/2022

Abstract

Low-code development platforms offer a streamlined approach to software development, utilizing visual interfaces and drag-and-drop utilities instead of traditional programming languages. This allows for faster application development and deployment, granting non-technical users access to tailored software solutions based on their unique needs. One key aspect of low-code development is the composition of model transformations, which enables developers to combine and reuse pre-existing models and components to create new applications. Model transformation composition refers to the process of combining multiple, simpler transformations to achieve a desired outcome. These compositions can range from converting one model format to another, extracting information, or manipulating data within a model. In various industries, such as software engineering, business process management, and product design, the use of model transformation composition can automate the creation of new software systems, improve business processes, and aid in product design.

The process of composing transformations presents a significant challenge to developers. Typically, smaller transformations are sourced from different and diverse sources and then manually combined, leading to a time-consuming and error-prone composition process. The application in low-code development platforms and the use of model transformation composition enable organizations to quickly and efficiently create and deploy business-critical applications based on the concept of model transformation and its composition.

This thesis aims to streamline the process by solving the issue of chaining various model transformations to create complex models. The solution involves externally combining simpler transformation steps to achieve this goal. The first step is identifying the different transformations that need to be chained. After identifying these transformations, selecting only those specific to the user is crucial, considering quality criteria such as the metamodels and transformations used. A search-based optimization approach utilizing model-driven techniques has been employed to tackle this selection problem. This process leads to optimizing the execution of the selected transformation chains, reducing the number of generated target elements and improving the overall execution time. Thus, this thesis is focused on finding a solution to the complex problem of composing transformations by utilizing search-based optimization and model-driven techniques to identify, select, and optimize the execution of the most efficient transformation chains according to user requirements.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Davide Di Ruscio, for their unwavering support, guidance, and encouragement throughout my PhD journey. His valuable insights and expertise have been instrumental in shaping my research and helping me to achieve my goals.

I would also like to thank my co-supervisors, Prof. Dr. Alfonso Pierantonio, for his valuable contributions to my research and for providing me with a diverse perspective on my work.

I am also grateful to the other professors and faculty members especially Prof. Dr. Ludovico Iovino, who have provided me with valuable feedback and support in writing research articles and performing experiments. I would also like to thank my research colleagues for their encouragement and for providing a sounding board for my ideas. My Ph.D. work has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement n° 813884.

Lastly, I would like to acknowledge the support of my family and friends, who have provided me with the love and encouragement needed to see this endeavor through to the end.

Thank you all for being a part of my PhD journey.

Contents

List of Figures

List of Tables

List of Abbreviations

MDE	**M**odel **D**riven **E**ngineering
LCDP	**L**ow **C**ode **D**evelopment **P**latform
BPMN	**B**usiness **P**rocess **M**odeling (and) **N**otations
MTC	**M**odel **T**ransformation **C**omposition
API	**A**pplication **P**rogramming **I**nterface
OCL	**O**bject **C**onstraint **L**anguage
ETL	**E**psilon **T**ransformation **L**anguage
EOL	**E**psilon **O**bject **L**anguage
EMG	**E**psilon **M**odel **G**eneration
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
SPML	**S**oftware **P**rocess **M**odeling **L**anguage
SPEM	**S**oftware **P**rocess **E**ngineering **M**etamodel
UML	**U**nified **M**odeling **Language**
ATL	**A**tlas **T**ransformation **L**anguage
MOMoT	**M**arrying Search-Based **O**ptimization and **Mo**del **T**ransformations
CMMI-DEV	**C**apability **M**aturity **M**odel **I**ntegration for **D**evelopment

Chapter 1

Introduction

Low-Code Development Platforms (LCDPs)[1] are visual environments that are being increasingly promoted by major IT players for supporting *citizen developers* to create software systems even if they lack programming background and knowledge [120]. One of the most prominent application domains for LCDPs is *process automation* [96]: citizen developers are provided with visual environments to specify workflows orchestrating sequential or even parallel consumption of services, each typically provided by external providers, which the used LCDP is able to connect. Thus, developers can specify processes e.g., to retrieve data from external data sources (e.g., calendar, sensors, and files stored in cloud services), to manipulate retrieved data. It can be achieved by means of the provided facilities or even by using external services and performing some aggregation and analysis according to rules defined by the languages provided by the platform. However, when complex workflows have to be specified, developers have to be aware of the possible service providers that the used LCDP is able to interact with and the manipulation ways that might be exploited to develop the desired process finally. LCDPs are software platforms that sit on the cloud and enable developers of different domain knowledge and technical expertise to develop fully-fledged applications ready for production [89]. Such applications are developed through model-driven engineering principles and take advantage of cloud infrastructures, automatic code generation, declarative and high level and graphical abstractions to develop entirely functioning applications [116]. These platforms capitalize on recent developments in cloud computing technologies and models such as Platform-as-a-service (PaaS), and proven software design patterns and architectures to ensure effective and efficient development, deployment and maintenance of the wanted application.

At the heart of low-code platforms, there are model-driven engineering (MDE) principles [21] that have been adopted in several engineering disciplines by relying on the automation, analysis, and abstraction possibilities enabled by the adoption of modelling and metamodelling [29]. Model driven engineering (MDE) elevates models as the first-class artefacts in the software development process. An MDE workflow typically involves several model management tasks such as model validation and model-to-model transformations. Model transformation is referred to as the heart and soul of MDE [103] and it can include model to model or model to text transformations (code generation). When software systems become complex, the workflow to build such applications can be specified using model transformation composition.

Model Transformation as a key concept in Model Driven Engineering (MDE) [26, 57] is applicable in low-code development platforms. It is a process of converting a model from one representation to another, often used in software development and engineering to bridge the gap between different modeling languages and tools. A model transformation manipulates and transform models by interpreting specified transformation rules [110], defined at the metamodel level, but executed at the model level. Some of the key features of low-code development that require model transformations are pre-built forms, reports and pages, interoperability with external sources such as

[1]Hereafter, the terms *low-code platforms* and *low-code development platforms* are used interchangeably and are abbreviated as *LCDPs*.

APIs[2], IFTTT[3], Zapier[4], etc. along with built-in workflows and converting report view from grid to Kanban to CSV, etc [96].

Many smaller and simpler model transformations can be chained together to realize a complex transformation ecosystem, when multiple transformations are available in a repository [22]. In such scenario, pre- and post-conditions of the transformations need to be ensured while the metamodels must be chained properly as per the compatibility of model transformations leading to the expected result according to the multiple transformations. The reusability and scalability of an applications can be achieved by composing model transformations that will preserve the syntactic and semantic properties of the composed smaller transformations. This helps in managing several workflows originated from different applications that may support heterogeneous platforms. The workflow engine in developing a scalable applications helps to manage goals in composing and reusing different models [92].

1.1 Challenges and motivations

Model transformation composition refers to the process of combining multiple model transformations together in order to achieve a desired outcome. One of the main challenges of model transformation composition is to ensure that the composed transformations are semantically consistent, meaning that the combined effect of the transformations preserves the intended meaning of the original model. Managing the complexity of the composed transformations, which can grow rapidly as the number of transformations increases is also an important requirement that needs to be considered. The developer must ensure that the composed transformations can be executed efficiently. Lastly, the model developer must preserve the properties of the original model by composed transformations, like consistency, integrity, completeness, etc.

Despite these challenges, there are enough motivation for model transformation composition that makes it practical to implement. The motivations are as follows. Reusing existing transformations to avoid having to reinvent the wheel is the most important uses to implement composition of model transformations. This would possibly create new functionality by combining existing transformations in new ways. Using composition to divide a large, complex problem into smaller, more manageable sub-problems is one of the primary reason to compose different transformations [109]. Also, automating the process of software development and maintenance by using model transformations to generate and update code is another application for model transformation compositions. Lastly, applying model transformations to different domains, where the properties of each domain are well-preserved, can also be achieved by combining them.

The scope of the thesis is invested mostly on reusing existing transformations and using several transformation criteria along with the involved metamodels that would determine the nature of the generated target model. Also, the thesis showcases that how a complex direct transformation can be replaced by the series of simpler transformations, given the requirement for the target model. Lastly, the thesis optimizes those transformations involved in the chaining that would minimize the execution time of a particular chain.

1.2 Research Activities

The general research objective is to compose and execute model transformations that enables the development of complex transformations by reusing and composing simpler and smaller ones

[2]Application Programming Interfaces
[3]https://ifttt.com/
[4]https://zapier.com/

[92]. There are three major activities performed in composing model transformation. They are the following. First is to identify the possible transformation chain by using cloud-based search engine [20] or through static analysis of the transformation language such as Epsilon [56]. Second major activity is to select transformation chain by estimating the optimal criteria such as transformation coverage, transformation complexity, number of transformation hops and model coverage [95]. Lastly, another activity involves optimizing a transformation chain by identifying the usage of the element that are propagated till the target model [94]. The optimization of the transformation chain also consider the use of equivalent operator in the epsilon transformation language (ETL).

1.2.1 Research Questions

The research questions that are addressed during the Ph.D. research work are as follows:

1. How to execute the set of transformations that are available in the transformation chain?
2. How can we automatically compose a set of model transformations to achieve the target model?
3. What are the quality criteria for characterizing the transformation chain (example, complexity, coverage, etc.) in term of the model and the metamodel used?
4. How can we achieve the quality criteria for characterizing transformation chain?
5. How can we optimize the execution of model transformation chain?

1.2.2 Achieved Outcome

The topic of the thesis is narrowed down to the aforementioned research questions. My Ph.D. work is divided into two major parts of research work. The first part is as follows. Initially, a feature survey has taken place that explains the usage of different characteristics for different low-code development platform [96]. Further, a business process and data-handling capacities for various low-code platforms are analyzed and compared with BPMN2.0 [93].

In the second part of the research work, there is a lot of coding involved. Here's a brief explanation of the process. The transformation chain is executed while preserving its semantic properties. It also involves identifying the available transformation chain for user-defined source and target metamodels, based on the availability of transformations and metamodels. After identifying the number of transformation chains, the best possible transformation chain is selected using optimization techniques based on various objectives such as complexities and coverage. Finally, optimization of the execution of the chosen transformation chain is achieved.

1.3 Structure of the thesis

Fig. 1.1 shows the structure of this document. This figure highlights the important parts of the thesis with different chapters. These chapters are discussed below.

Chapter 2 - State of the art: It provides literature on the following topics. Section 2.1 presents the literature on model transformation composition whereas section 2.2 presents quality criteria in model transformation. Section 2.3 introduces the search-based techniques in model transformation while section 2.4 presents the optimization involved in the model transformation chain.

Chapter 3 - Understanding low-code development platforms: It provides the overall usages of low-code development platforms and elaborates on the features involved in the low-code platforms. Section 3.1 presents the overall structure involved in explaining the low-code platform

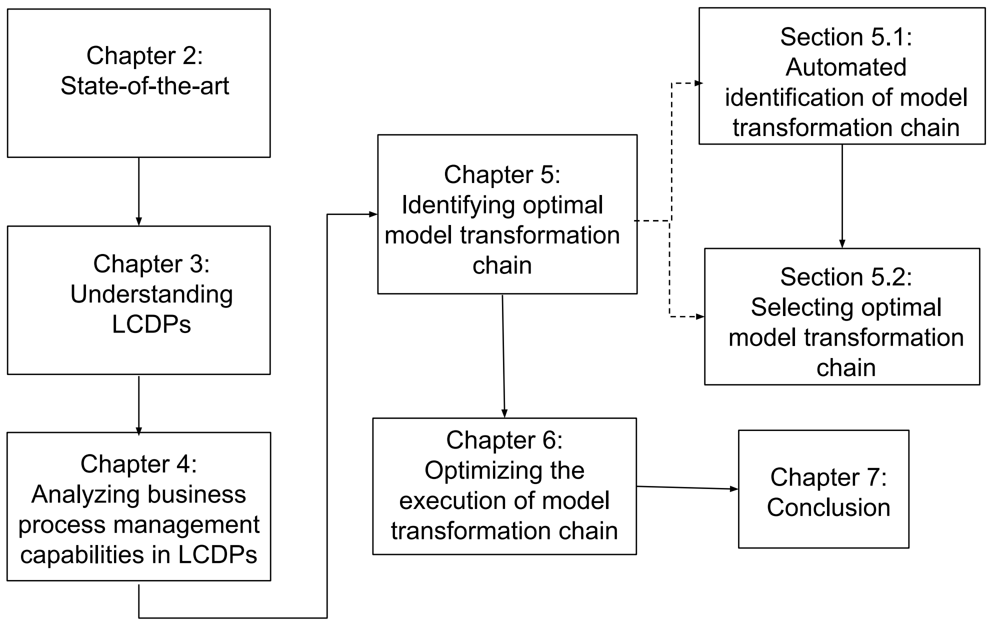

Chapter 2: State-of-the-art
Chapter 3: Understanding LCDPs
Chapter 4: Analyzing business process management capabilities in LCDPs
Chapter 5: Identifying optimal model transformation chain
Section 5.1: Automated identification of model transformation chain
Section 5.2: Selecting optimal model transformation chain
Chapter 6: Optimizing the execution of model transformation chain
Chapter 7: Conclusion

FIGURE 1.1: Structure of the thesis

while section 3.2 elaborates on the major components of the low-code platform. Consequently, section 3.3 presents the development process of a low-code development platform. Section 3.4 presents a brief description for various low-code development platforms. The feature is explained and categorized the eight considered low-code platforms by creating feature taxonomy is explained in section 3.5. Section 3.6 presents the comparison of the features and capabilities of low-code platform while explaining the additional aspects when comparing low-code platforms. Lastly, section 3.7 presents the experience report of using a low-code platform while developing a software application.

Chapter 4 - Analyzing business process management capabilities of LCDPs: It provides an elaborate analysis of low-code development platforms with a standard business process tool such as BPMN. This chapter also explains the data-handling capabilities used in various low-code development platforms. Section 4.1 presents the description of process modeling with BPMN with a motivating example. Section 4.2 presents a short survey for various process modelling languages while focusing on various quality criteria of process modelling languages. Section 4.3 discusses the process modelling and data-handling constructs in the eight considered low-code development platforms. Section 4.4 presents the discussion of various LCDPs with BPMN modelling constructs. Lastly, section 4.5 presents the discussion of various LCDPs with respect to BPMN quality criteria.

Chapter 5 - Identifying optimal model transformation chains: It provides the approaches for identifying model transformation chains and selecting the optimal model transformation chain. Section 5.1 presents the approach used for identifying model transformation chain and the output identified from a repository of metamodels and transformations. Section 5.2 presents the background and motivating dummy example of selecting the optimal chain. This chapter focuses on the search-based approaches to select the optimal transformation chain. This chapter presents the experiments done on chain selection given on a specific problem configuration, threats to validity along with the discussion on the achieved output done in the experiments.

Chapter 6 - Optimizing the execution of model transformation chains: This chapter focuses on optimizing the execution of model transformation chains. Section 6.1 presents the background and the motivating example of optimizing the transformation chain. The proposed approach for optimizing the transformation chain execution is presented in Section 6.2. Section 6.3 presents the experimental evaluation of the approach along with the results obtained in terms of execution time and target-generated elements of the transformation chain. Lastly, section 6.4 presents the threats to the validity of the proposed approach.

Chapter 7 - Conclusion: This chapter focuses on outlining the contribution of the thesis while presenting the different publications that are either published or are under submission. Also, developed tools and future work are mentioned in this chapter.

Chapter 2

State-of-the-art

This chapter is dedicated to the literature on the topics of the research. In particular, an overview of existing work related to the following subjects is given: model transformation composition, quality criteria used in model transformations, search-based techniques involved in model transformations and optimization of various model transformation chains.

2.1 Model Transformation composition

In Lucio et al.(2013) [68], explicit modeling activities are used and analysed by combining Formalism Transformation Graph and Process Model (FTG+PM) to find out the target model or *intent* of the model transformation chain. This *intent* are attached as annotations for each available transformations that will compose each properties of a transformations to achieve the desired outcome.

Kuster et al.(2009) [65] describe an incremental development of model transformation chain based on the automated testing. These testing framework would improve the quality of transformation chain incrementally that also allows a developer to change an individual transformation without affecting the chain.

Alvarez et al.(2013) [7] present a tool called MTC Flow that allows developers to design, develop, test and deploy model transformation chains. MTC Flow uses graphical domain specific languages (DSLs) for defining workflow models for transformation chain that are independent to the technologies that support the transformations. Also, this tool is interoperable to any transformation or validation technologies that describes the multiple transformation chains by designing their workflow model.

In Aranega et al.(2012) [13], feature models (based on separation of concern) are proposed to classify bigger model transformations. These feature models helps to automate the valid set of model transformations and generate the executable chain of model transformations that gives the desired output model.

The above mentioned research papers explain the concept of model transformation composition that enables developers to combine and reuse the pre-existing models and components to build new applications. The process of combining multiple, simpler transformations to achieve a desired outcome is done by composing different model transformations. These compositions can range from transforming one model format to another, extracting information, or manipulating data within a model. The importance of model transformation composition is also highlighted in the above mentioned papers. The transformation composition is used in various industries such as automation in software systems, improving business processes, and product design. However, the automation of the execution of transformation chains are not done previously and we achieved the automatic execution of the transformation chain and the optimizing their execution in terms of memory space and execution time.

2.2 Quality criteria in model transformations

Syriani et al.(2012) [107] elaborate on the design pattern in the context of model transformation that satisfy quality criteria identified before the execution of the transformation. The quality criteria identified to assess and validate the model transformation design pattern are correctness, reusability, efficiency, reliability, maintainability and interoperability. The verification and the validation is done on the design patterns allows to assess whether the cataloged design patterns are complete with respect to the quality criteria. This helps to detect the bad design and improve the design pattern of a model transformation.

In Selim et al.(2012) [102], transformation testing is used to estimate the quality criteria of the model transformation. Some of the estimated quality criteria for transformation testing are metamodel coverage, input contract coverage of the model transformation, etc. These quality criteria are calculated using mutation analysis which computes the value of a quality criteria of the original model and compare it with the value of quality criteria when the model is mutated or changed.

Baurer et al.(2011) [23] present a coverage analysis approach to measure the test suite quality for model transformation chains. Their approach combines different coverage criteria such as class coverage, attribute coverage, association coverage, feature coverage and transformation contract coverage. The combination of these coverage gives a detailed coverage information that are used to identify missing and redundant test cases of model transformation or model transformation chains.

In Ergin et al.(2013) [42], a new model transformation design pattern is introduced that improves quality in model transformation. The design pattern focuses on three quality metrics of the transformation. They are the number of rule applications, the size of the rule and the number of auxiliary elements. These three metrics are related to the efficiency quality criteria and is therefore, improvements in these metrics would lead to the reduction of time complexity. This paper finds out that the normal usage of these metrics would lead to the quadratic time complexity while the improved solution would lead to linear time complexity.

Mkaouer et al.(2014) [76] elaborate on the objectives of the model transformation which provide rules that generates the target model without any error and to minimize the complexity of the transformation rules (by reducing number of rules and bindings in the same rule), while maximizing the quality of the target models. This paper focuses on the transformation mechanism as a multi-objective problem to find the best rules that maximize the target model quality and minimize the rule complexity. The quality of the transformations rules and bindings are iteratively improved by using the multi-objective optimization process. The objectives are the number of rules and matching metamodels in each rule and assessing the quality of generated target models using a set of quality metrics. Optimization algorithm such as NSGA-II is used to automatically generate the best transformation rules satisfying the two conflicting objectives. By achieving the best possible solution with two conflicting objectives, the paper claims to provide a well-organized target models with a minimal set of rules.

In Basciani et al.(2018) [19], two quality criteria such as transformation coverage and information loss considered in a model transformation chain scenario where multiple chains are possible between the source and the target model. A customized Dijkstra algorithm is used to individually consider the two criteria that consider the best chain. Since, the information loss is considered on the model generated, the paper claims that information loss is a better quality criteria as compared to the transformation coverage which is considered by calculating the static element in the metamodel and the transformation without considering the generated models.

The above mentioned papers in the subsection explain the quality criteria in model transformations that are used to evaluate the correctness and effectiveness of model transformations. The quality criteria can be broadly classified into categories: intrinsic and extrinsic. Intrinsic criteria are related to the transformation itself such as coverage criteria, rule size, etc. Whereas, extrinsic criteria are related to the context in which the transformation is used such as reusability, reliability, etc. The thesis is focussed on intrinsic quality criteria of the model transformation such as the optimal model transformation composition can be identified.

2.3 Search-based approaches in model transformations

In Kessentini et al.(2008) [60], the transformation technique is framed as a combinatorial optimization problem where the end goal is to find a better transformation that starts from a small set of available examples. The search-based model transformation by example is also further elaborated in the paper [61]. This approach is called MOdel Transformation as Optimization by Example (MOTOE) combines transformation blocks extracted from examples in order to generate a target model. A modified version of particle swarm optimization (PSO) is used where different transformation solutions are modelled as particle which exchange transformation blocks to converge to achieve an optimal transformation solution.

Fleck et al.(2017) [46] identify the problem of modularizing model transformation program and use it as a model in an automated search-based approach. The application and the execution of the problem is managed by the search framework that combines an in-place transformation language (in Henshin) and uses a search-based algorithm framework. This calculates the pareto-optimal solution based on four objectives or quality attributes. The four objectives are the number of modules in the transformation, the difference between the lowest and highest number of responsibility in a module, the cohesion ratio and the coupling ratio. This approach uses MOMoT framework [44] that can model a problem and by using in-place transformation and search-based algorithm (such as NSGA-II, NSGA-III, etc), a pareto-optimal solution is found out based on minimization or maximization of the quality objectives.

Sahin et al.(2015) [98] propose to handle model transformation testing as a bi-level optimization problem which combines the generation of test cases with mutation testing. This paper divides the problem into two parts: the upper level and the lower level. The upper level problem generates a set of test cases that use to maximize the coverage of metamodels used and the errors introduced by the lower level problem to the transformation rules. This bi-level formulation of the problem provides a statistical analysis of the obtained result that shows the competitiveness and the outperforming of the proposed approach as compared to the precision over co-evolution and non-search-based methods.

The discussed papers explain how search-based optimization techniques are used to select and optimize transformation chains according to user requirements. The aforementioned papers highlight the search-based approach as a means to tackle the issue of chaining various model transformations to create the desired output models. This sub-chapter also focuses on the use of search-based optimization and model-driven techniques that helps to find out an efficient solution to the complex problem of composing various transformations. We have used search-based and model-driven approaches mentioned in paper [44]. The discussed paper helps to coherently joined the model driven approach of framing the transformation chain selection problem and running the search-based algorithm designed in the MOMoT framework.

2.4 Optimization of model transformation chains

Cabot et al. (2006) [31] propose a metric to measure the complexity of the OCL expression. The metric is based on the syntactic structure of the expressions (number of referred attributes, number of navigation, etc.) and on the constructs used in their definition (such as the number of `forAll` and `select` iterators). This traversal of each expression to determine number of objects involved in calculating the expressions aims to give a precise complexity of the execution of an OCL expression.

In Cuadrado et al.(2020) [101], authors have proposed an approach for parallel execution of model transformations along with some optimizations both at the transformation level and at OCL[1] expression level using static analysis. Firstly, Ordering of the matched rules is done to identify the matched rule. Second, the foot-printing of the model transformations to filter the model elements is done by only keeping the ones that are matched with any rule. Third, handling bindings to be resolved at compile-time to speedup the execution at run-time. Fourth, implementing trace links reduction by only keep trace of those links that are used during the transformation. This approach doesn't take chain optimization into account.

An approach in Le Calvar et al.(2019) [67] compiles a subset of ATL code to generate efficient Java code. A similar approach to enable incremental execution of model transformations in [86] uses partial evaluation to pre-compute a part of model transformation.

Viatra [115] provides an incremental engine based on RETE algorithm [125]. The incremental engine of Viatra computes the pattern matches and caches them, thus, executing the transformations efficiently. But due to higher caching of the memory consumption, the user gets a faster execution at the cost of more memory consumption.

The papers above highlight the difficulties that modellers and developers face when composing model transformations. The challenges stem from the varying sizes and diverse sources of the model transformations, which must be manually composed in a time-consuming and error-prone manner. In this thesis, we have improved the execution of the model transformation chain by reducing the unnecessary target modeling elements and transformation rules. This optimization leads to faster execution time and more efficient memory usage while generating the final output of the transformation chain.

[1]Object Constraint Language

Chapter 3

Understanding Low-Code Development Platforms

By using low-code development platforms (LCDPs), citizen developers can build their software application without the help of traditional developers that were earlier involved along with the full-stack development of fully operational applications. Thus, low-code developers can focus on the business logic of the application being specified rather than dealing with unnecessary details related to setting up of the needed infrastructures, managing data integrity across different environments, and enhancing the robustness of the system. Bug fixing and application scalability and extensibility are also made easy, fast and maintainable in these platforms by the use of high-level abstractions and models [10].

In this chapter, a technical survey is provided to distil the relevant functionalities provided by different LCDPs and accurately organize them. In particular, eight major LCDPs have been analyzed to provide potential decision-makers and adopters with objective elements that can be considered when educated selections and considerations have to be performed. The contributions of this chapter are summarized as follows:

- Identification and organization of relevant features characterizing different low-code development platforms;
- Comparison of relevant low-code development platforms based on the identified features;
- Presentation of a short experience report related to the adoption of LCDPs for developing a simple benchmark application.

3.1 A bird-eye view of low-code development platforms

From an architectural point of view, LCDPs consist of four main layers, as shown in Fig. 3.1. The top layer (see `Application Layer`) consists of the graphical environment that users directly interact to specify their applications. The tool boxes and widgets used to build the user interface of the specified application are part of this layer. It also defines authentication and authorization mechanisms to be applied to the specified artefacts. Through the modelling constructs made available at this layer, users can specify the behaviour of the application being developed. For instance, users can specify how to retrieve data from external data sources (e.g., spreadsheets, calendars, sensors, and files stored in cloud services), how to manipulate them by using platform facilities or utilising external services, how to aggregate such data according to defined rules, and how to analyze them. To this end, the `Service Integration Layer` is exploited to connect with different services by using corresponding APIs and authentication mechanisms.

A dedicated data integration layer permits to operate and homogeneously manipulate data even if heterogeneous sources are involved. To this end, the `Data Integration Layer` is concerned with

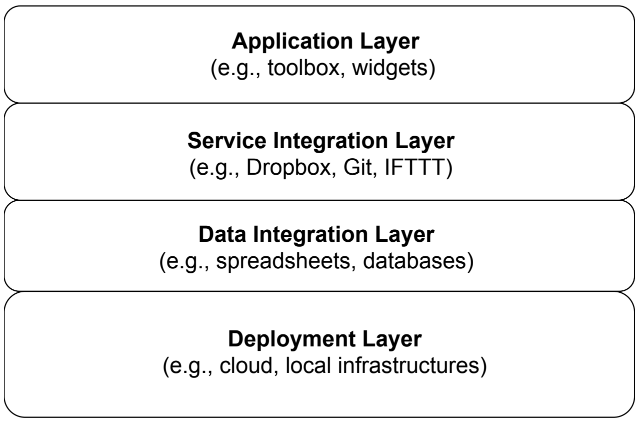

Application Layer
(e.g., toolbox, widgets)
Service Integration Layer
(e.g., Dropbox, Git, IFTTT)
Data Integration Layer
(e.g., spreadsheets, databases)
Deployment Layer
(e.g., cloud, local infrastructures)

FIGURE 3.1: Layered architecture of low-code development platforms

data integration with different data sources. Depending on the used LCDP, the developed application can be deployed on dedicated cloud infrastructures or on-premise environments (`Deployment Layer`). Note that the containerization and orchestration of applications are handled at this layer together with other continuous integration and deployment facilities that collaborate with the `Service Integration Layer`.

3.2 Main components of low-code development platforms

By expanding the layered architecture shown in Fig. 3.1, the distinct components building any low-code development platform are depicted in Fig. 3.2 and they can be grouped into three tiers. The first tier is made of the application modeler, the second tier is concerned with the server side and its various functionalities, and the third tier is concerned with external services that are integrated with the platform. The arrows in Fig. 3.2 represent possible interactions that might occur among entities belonging to different tiers. The lines shown in the middle tier represents the main components building up the platform infrastructure.

As previously mentioned, modelers are provided with an *application modeler* enabling the specification of applications through provided modeling constructs and abstractions. Once the application model has been finalized, it can be sent to the platform back-end for further analysis and manipulations including the generation of the full-fledged application, which is tested and ready to be deployed on the cloud.

Figure 3.3 shows the application modeler of Mendix [73] at work. The right-hand side of the environment contains the widgets that modelers can use to define applications, as shown in the central part of the environment. The left-hand side of the figure shows an overview of the modeled system in terms of, e.g., the elements in the domain model, and the navigation model linking all the different specified pages. The application modeler also permits to run the system locally before deploying it. To this end, as shown in Fig. 3.2, the middle tier takes the application model received from the application modeler and performs model management operations including code generations and optimizations by also considering the involved services including database systems, micro-services, APIs connectors, model repositories of reusable artifacts, and collaboration means [80].

Concerning *database servers*, they can be both SQL and NoSQL. In any case, the application users and developers are not concerned about the type of employed database or mechanisms ensuring data integrity or query optimizations. More in general, the developer is not concerned about low-level architecture details of the developed application. All the needed *micro-services* are created,

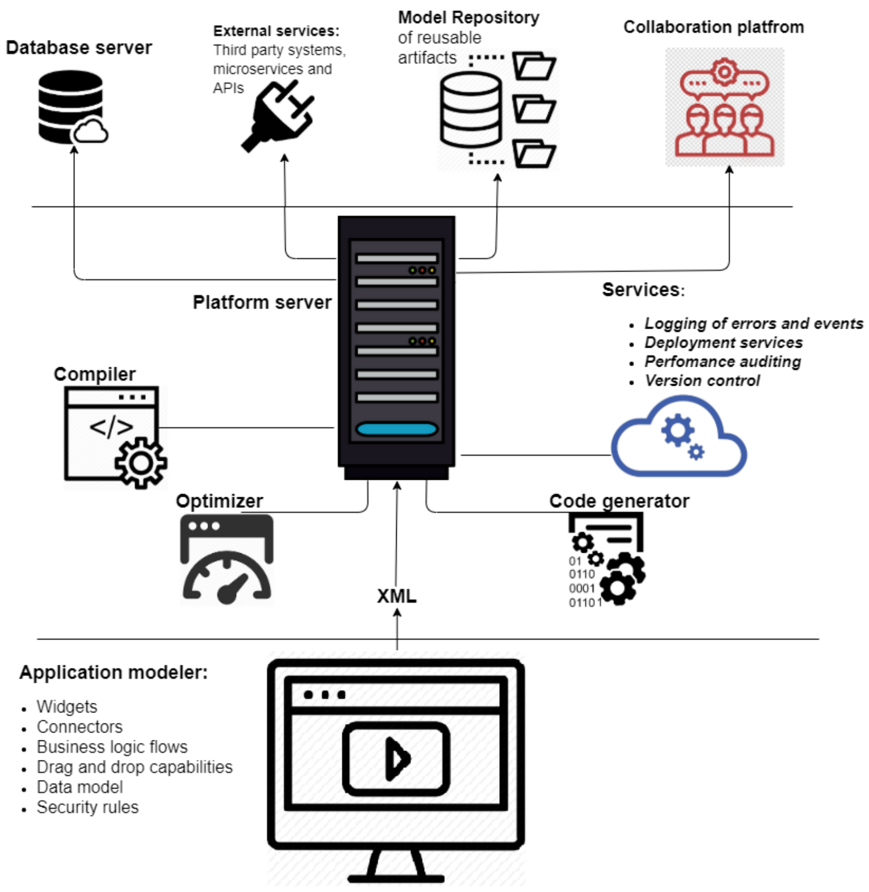


FIGURE 3.2: Main components of low-code development platforms

orchestrated and managed in the back-end without user intervention. Although the developer is provided with the environment where she can interact with external *APIs*, there are specific connectors in charge of consuming these APIs in the back-end. Thus, developers are relieved from the responsibility of manually managing technical aspects like authentication, load balance, business logic consistency, data integrity and security.

Low-code development platforms can also provide developers with repositories that can store reusable modeling artifacts by taking care of version control tasks. To support *collaborative development* activities, LCDPs include facilities supporting development methodologies like agile, kanban, and scrum. Thus, modelers can easily visualize the application development process, define tasks, sprints and deal with changes as soon as customers require them and collaborate with other stakeholders.

3.3 Development process in LCDPs

The typical phases that are performed when developing applications by means of LCDPs can be summarized as follows.

1. *Data modeling* - usually, this is the first step taken; users make us of a visual interfaces to configure the data schema of the application being developed by creating entities, establishing relationships, defining constraints and dependencies generally through drag-and-drop facilities. A simple data model defined in Mendix is shown in Fig. 3.4.

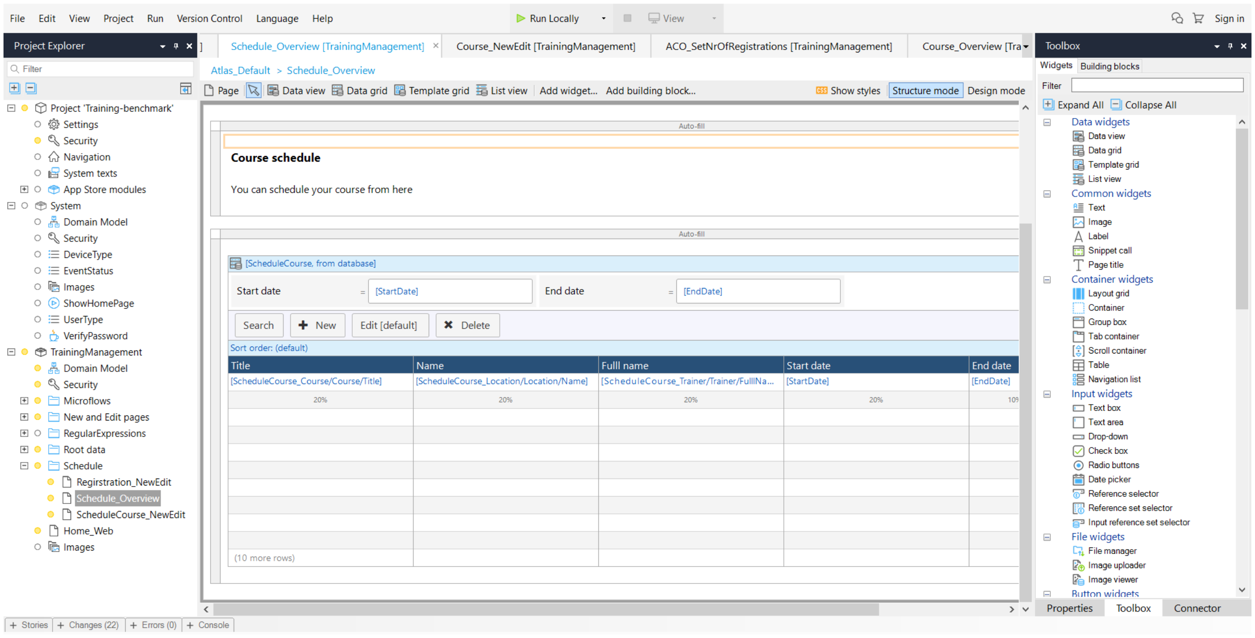


FIGURE 3.3: The application modeler of Mendix at work

2. *User interface definition* - secondly, the user configures forms and pages (e.g., see Fig. 3.3) used to define the application views, and later define and manage user roles and security mechanisms across at least entities, components, forms, and pages. It is here that drag-and-drop capabilities play a significant role to speed up development and render the different views quickly.

3. *Specification of business logic rules and workflows* - Third, the user might need to manage workflows amongst various forms or pages requiring different operations on the interface components. Such operations can be implemented in terms of visual-based workflows and to this end, BPMN-like notations can be employed as, e.g., shown in Fig. 3.5.

4. *Integration of external services via third-party APIs* - Fourth, LCDPs can provide means to consume external services via integration of different APIs. Investigating the documentation is necessary to understand the form and structure of the data that can be consumed by the adopted platform.

5. *Application Deployment* - In most platforms, it is possible to quickly preview the developed application and deploy it with few clicks.

3.4 An overview of representative low-code development platforms

This section presents an overview of eight low-code development platforms that have been considered as leaders in the related markets from recent Gartner [116] and Forrester [91] reports. These eight low-code platforms are assumed to be representative platforms for the benefit of our analysis that encompasses diverse feature capabilities mentioned in Table 3.1.

OutSystem [80] is a low-code development platform that allows developing desktop and mobile applications, which can run in the cloud or in local infrastructures. It provides inbuilt features which enable to publish an application via a URL with a single button click. OutSystems has two significant components. First, it has an intermediate Studio for database connection through .NET or Java and secondly, it has a service studio to specify the behaviour of the application being

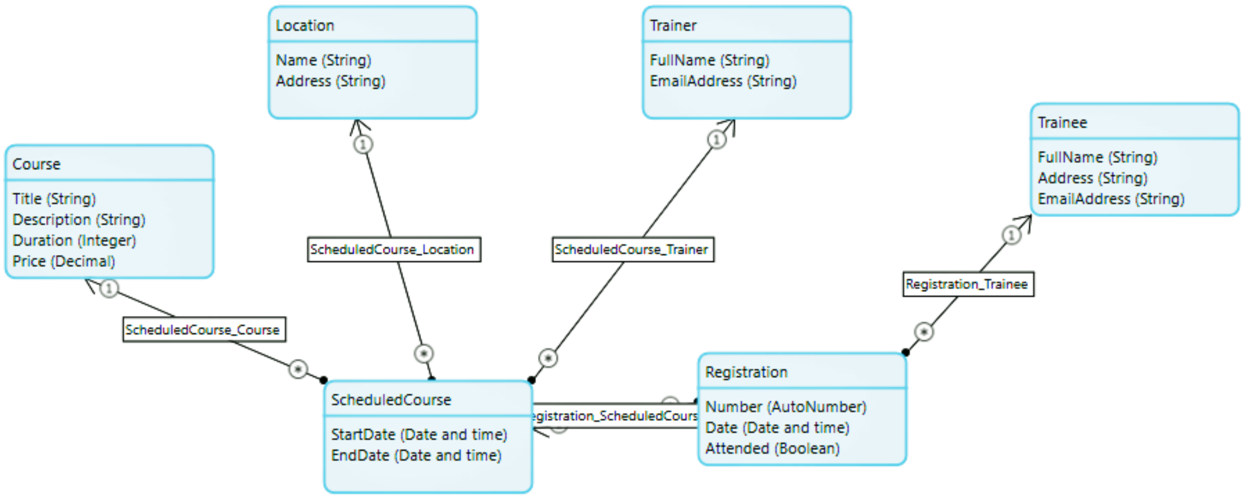


FIGURE 3.4: A simple data model defined in Mendix

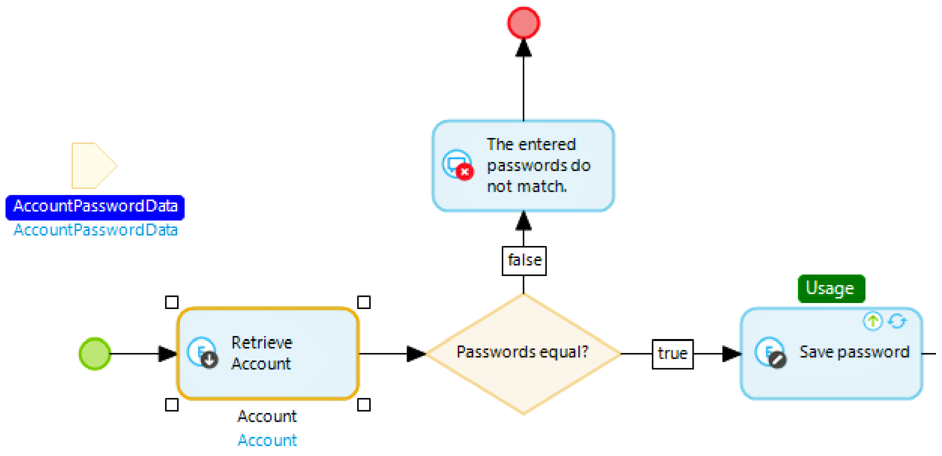


FIGURE 3.5: A simple logic defined in Mendix

developed. Some of the supported applications in this platform are billing systems, CRMs[1], ERPs [2], extensions of existing ERP solutions, operational dashboards and business intelligence.

Mendix [73] is a low-code development platform that does not require any code writing and all features can be accessed through drag-and-drop capabilities while collaborating in real-time with peers. There is a visual development tool that helps to reuse various components to fasten the development process from the data model setup to the definition of user interfaces. Users can create some context-aware apps with pre-built connectors, including those for the IoT, machine learning, and cognitive services. Mendix is compatible with Docker[3] and Kubernetes[4], and it has several application templates that one can use as starting points. Mendix's Solution Gallery[5] is an additional resource that permits users to start from already developed solutions, and that might be already enough to satisfy the requirements of interest.

Zoho Creator [127] offers drag-and-drop facilities to make the development of forms, pages and dashboards easy. The provided user interface supports web design where the layout of the page

[1]Customer Relationship Managements
[2]Enterprise Resource Plannings
[3]https://www.docker.com/
[4]https://kubernetes.io/
[5]https://www.mendix.com/solutions/

reflects the resolution of the screen of the user (e.g., in the case of mobile or desktop applications). It also offers integration with other Zoho apps and other Salesforce[6] connectors. Customized workflows are essential features of Zoho Creator.

Microsoft PowerApps [75] supports drag-and-drop facilities and provides users with a collection of templates which allows reuse of already developed artifacts. A user can follow model-driven or canvas approaches while building applications. PowerApps integrates with many services in the Microsoft ecosystem such as Excel, Azure database[7] or similar connectors to legacy systems.

Google App Maker [52] allows organizations to create and publish custom enterprise applications on the platform powered by G Suite[8]. It utilizes a cloud-based development environment with advanced features such as in-built templates, drag-and-drop user interfaces, database editors, and file management facilities used while building an application. To build an extensive user experience, it uses standard languages such as HTML, JavaScript, and CSS.

Kissflow [62] is a workflow automation software platform based on the cloud to help users to create and modify automated enterprise applications. Its main targets are small business applications with complete functional features which are essential for internal use, and human-centred workflows such as sales enquiry, purchase request, purchase catalogue, software directory, and sales pipeline. It supports integration with third-party APIs, including Zapier[9], Dropbox[10], IFTTT[11], and Office 365[12].

Salesforce App Cloud [99] helps developers to build and publish cloud-based applications which are safe and scalable without considering the underlying technological stacks. It exhibits out-of-the-box tools and operations for automation by integrating them with external services. Some of the peculiar features are the extensive AppExchange marketplace[13] consisting of pre-built applications and components, reusable objects and elements, drag-and-drop process builder, and inbuilt kanban boards.

Appian [12] is one of the oldest low-code platform, which permits to create mobile and Web applications through a customized tool, built-in team collaboration means, task management, and social intranet. Appian comes with a decision engine which is useful for modeling complex logic.

3.5 Taxonomy

In this section we introduce preparatory terms, which can facilitate the selection and comparison of different LCDPs. The features are derived by examining the requirements in building an application along with the capabilities that a low-code platform could offer in achieving the making of an application. In particular, by analyzing the low-code development platforms described in the previous section, we identified and modeled their variability and commonalities. Our results are documented using feature diagrams [34], which are a common notation in domain analysis [33]. Fig. 3.6 shows the top-level feature diagram, where each sub-node represents a major point of variation. Table 3.1 gives details about the taxonomy described in the following.

- *Graphical user interface:* This group of features represents the provided functionalities available in the front-end of the considered platform to support customer interactions. Examples

[6]https://www.salesforce.com/it/
[7]https://azure.microsoft.com
[8]https://gsuite.google.com/
[9]https://zapier.com
[10]https://www.dropbox.com/
[11]https://ifttt.com/
[12]https://products.office.com/it-it/home
[13]https://appexchange.salesforce.com/

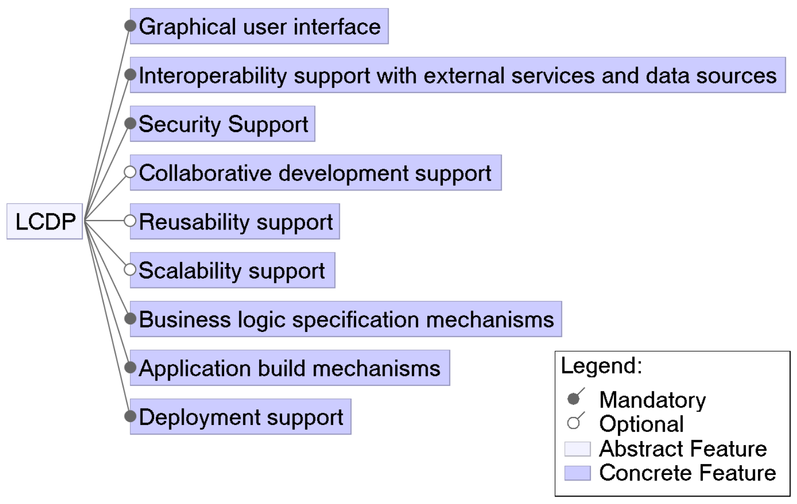

Graphical user interface
Interoperability support with external services and data sources
Security Support
Collaborative development support
LCDP
Reusability support
Scalability support
Business logic specification mechanisms
Application build mechanisms
Deployment support
Legend:
Mandatory
Optional
Abstract Feature
Concrete Feature

FIGURE 3.6: Feature diagram representing the top-level areas of variation for LCDPs

of features included in such a group are drag-and-drop tools, forms, and advanced reporting means.

- *Interoperability support with external services and data sources*: This group of features is related to the possibility of interacting with external services such as Dropbox, Zapier, Sharepoint, and Office 365. Also, connection possibilities with different data sources to build forms and reports are included in such a group.
- *Security support*: The features in this group are related to the security aspects of the applications that are developed by means of the employed platform. The features included in such a group include authentication mechanisms, adopted security protocols, and user access control infrastructures.
- *Collaborative development support*: Such a group is related to the collaboration models (e.g., online and off-line) that are put in place to support the collaborative specification of applications among developers that are located in different locations.
- *Reusability support*: It is related to the mechanisms employed by each platform to enable the reuse of already developed artifacts. Examples of reusability mechanisms are pre-defined templates, pre-built dashboards, and built-in forms or reports.
- *Scalability support*: Such a group of feature permits developers to scale up applications according to different dimensions like the number of manageable active users, data traffic, and storage capability that a given application can handle.
- *Business logic specification mechanisms*: It refers to the provided means to specify the business logic of the application being modeled. The possibilities included in such a group are business rules engine, graphical workflow editor, and API support that allows one application to communicate with other application(s). Business logic can be implemented by using one or more API call(s).
- *Application build mechanisms*: It refers to the ways the specified application is built, i.e., by employing code generation techniques or through models at run-time approaches. In the former, the source code of the modeled application is generated from the specified models and subsequently deployed. In the latter, the specified models are interpreted and used to manage the run-time execution of the application.

- *Deployment support*: The features included in such a group are related to the available mechanisms for deploying the modeled application. For instance, once the system has been specified and built, it can be published in different app stores and deployed in local or cloud infrastructures.

In addition to the top-level features shown in Fig. 3.6, LCDPs can be classified also with respect to the *Kinds of supported applications*. In particular, each LCDP can specifically support the development of one or more kinds of applications including Web portals, business process automation systems, and quality management applications.

TABLE 3.1: Taxonomy for Low-Code Development Platforms

Feature	**Description**
Graphical user interface	
Drag-and-drop designer responses,	This feature enhances the user experience by permitting to drag all the items involved in making an app including actions, connections, etc.
Point and click approach	This is similar to the drag-and-drop feature except it involves pointing on the item and clicking on the interface rather than dragging and dropping the item.
Pre-built forms/reports	This is off-the-shelf and most common reusable editable forms or reports that a user can use when developing an application.
Pre-built dashboards	This is off-the-shelf and most common dashboards that a user can use when developing an application.
Forms	This feature helps in creating a better user interface and user experience when developing applications. A form includes dashboards, custom forms, surveys, checklists, etc. which could be useful to enhance the usability of the application being developed.
Progress tracking	This features helps collaborators to combine their work and track the development progress of the application.
Advanced Reporting	This features enables the user to obtain a graphical reporting of the application usage. The graphical reporting includes graphs, tables, charts, etc.
Built-in workflows	This feature helps to concentrate the most common reusable workflows when creating applications.
Configurable workflows	Besides built-in workflows, the user should be able to customize workflows according to their needs.
Interoperability support	
Interoperability services	This feature is one of the most important features to incorporate different services and platforms including that of Microsoft, Google, etc. It also includes the interoperability possibilities among different low-code platforms.
Connecting data sources	This features connects the application with data sources such as Microsoft Excel, Access and other relational databases such as Microsoft SQL, Azure and other non-relational databases such as MongoDB.
Security Support	
Application security application,	This feature enables the security mechanism of an application which involves confidentiality, integrity and availability of an if and when required.
Platform security	The security and roles management is a key part in developing an application so that the confidentiality, integrity and authentication (CIA) can be ensured at the platform level.
Collaborative dev support	
Off-line collaboration to	Different developers can collaborate on the specification of the same application. They work off-line locally and then they commit a remote server their changes, which need to be properly merged.
On-line collaboration	Different developers collaborate concurrently on the specification of the same application. Conflicts are managed at run-time.
Reusability support	
Built-in workflows	This feature helps to concentrate the most common reusable workflows in creating an application.
Pre-built forms/reports	This is off-the-shelf and most common reusable editable forms or reports that a user might want to employ when developing an application.
Pre-built dashboards	This is off-the-shelf and most common dashboards that a user might want to employ when developing an application.
Scalability	
Scalability on no. of users	This features enables the application to scale-up with respect to the number of active users that are using that application at the same time.
Scalability on data traffic	This features enables the application to scale-up with respect to the volume of data traffic that are allowed by that application in a particular time.
Scalability on data storage	This features enables the application to scale-up with respect to the data storage capacity of that application.
Business logic spec mechanisms	
Business rules engine	This feature helps in executing one or more business rules that help in managing data according to user's requirements.
Graphical workflow editor	This feature helps to specify one or more business rules in a graphical manner.
AI enabled business logic	This is an important feature which uses Artificial Intelligence in learning the behaviour of an attributes and replicate those behaviours according to learning mechanisms.
Application build mechanisms	
Code generation	According to this feature, the source code of the modeled application is generated and subsequently deployed before its execution.
Models at run-time	The model of the specified application is interpreted and used at run-time during the execution of the modeled application without performing any code generation phase.
Deployment support	
Deployment on cloud	This features enables an application to be deployed online in a cloud infrastructure when the application is ready to deployed and used.
Deployment on local infra	This features enables an application to be deployed locally on the user organization's infrastructure when the application is ready to be deployed and used.
Kinds of supported applications	
Event monitoring	This kind of applications involves the process of collecting data, analyzing the event that can be caused by the data, and signalling any events occurring on the data to the user.
Process automation	This kind of applications focuses on automating complex processes, such as workflows, which can takes place with minimal human intervention.
Approval process control user.	This kind of applications consists of processes of creating and managing work approvals depending on the authorization of the For example, payment tasks should be managed by the approval of authorized personnel only.
Escalation management out aspects	This kind of applications are in the domain of customer service and focuses on the management of user viewpoints that filter that are not under the user competences.
Inventory management	This kind of applications is for monitoring the inflow and outflow of goods and manages the right amount of goods to be stored.
Quality management	This kind of applications is for managing the quality of software projects, e.g., by focusing on planning, assurance, control and improvements of quality factors.
Workflow management	This kind of applications is defined as sequences of tasks to be performed and monitored during their execution, e.g., to check the performance and correctness of the overall workflow.

3.6 Comparing relevant LCDPs

In this section, we make use of the taxonomy previously presented to compare the eight low-code development platforms overviewed in Sec. 3.4. Table 3.2 shows the outcome of the performed comparison by showing the corresponding supported features for each platform. The data shown in Table 3.2 are mainly obtained by considering the official resources of each platform as referenced by [80],[73], [127],[75],[52],[62],[99], [12], and by considering the experience we gained during the development of a benchmark application as discussed in the next section.

TABLE 3.2: Comparison of analysed low-code development platforms

Feature	**OutSystems**	**Mendix**	**Zoho Creator**	**MS PowerApp**	**App Maker**	**Kissflow**	**Salesforce App**	**Appian**
Graphical user interface								
Drag-and-drop designer	✓	✓	✓		✓	✓	✓	✓
Point and click approach				✓				
Pre-built forms/reports	✓	✓	✓	✓	✓	✓	✓	✓
Pre-built dashboards	✓		✓	✓		✓	✓	
Forms			✓	✓				
Progress tracking	✓	✓	✓	✓	✓	✓	✓	✓
Advanced reporting						✓		
Built-in workflows			✓			✓	✓	
Configurable workflows			✓			✓	✓	
Interoperability support								
Interoperability with external service	✓	✓	✓	✓		✓	✓	✓
Connection with data sources	✓	✓	✓	✓	✓	✓	✓	✓
Security Support								
Application security	✓	✓	✓	✓	✓	✓	✓	✓
Platform security	✓	✓	✓	✓	✓	✓	✓	✓
Collaborative development support								
Off-line collaboration	✓	✓	✓	✓	✓	✓	✓	✓
On-line collaboration	✓	✓			✓	✓	✓	✓
Reusability support								
Built-in workflows			✓			✓	✓	
Pre-built forms/reports	✓	✓	✓	✓	✓	✓	✓	✓
Pre-built dashboards	✓		✓	✓		✓	✓	
Scalability								
Scalability on number of users	✓	✓	✓	✓	✓	✓	✓	✓
Scalability on data traffic	✓	✓	✓	✓	✓	✓	✓	
Scalability on data storage	✓	✓	✓	✓	✓	✓	✓	
Business logic specification mechanisms								
Business rules engine	✓	✓	✓	✓	✓	✓	✓	✓
Graphical workflow editor	✓	✓				✓	✓	
AI enabled business logic	✓					✓	✓	✓
Application build mechanisms								
Code generation	✓							
Models at run-time		✓	✓	✓	✓	✓	✓	✓
Deployment support								
Deployment on cloud	✓	✓	✓	✓	✓	✓	✓	✓
Deployment on local infrastructures	✓	✓					✓	✓
Kinds of supported applications								
Event monitoring	✓	✓	✓	✓	✓	✓	✓	✓
Process automation	✓		✓	✓	✓	✓		✓
Approval process control					✓			
Escalation management						✓		
Inventory management	✓	✓	✓	✓	✓	✓	✓	✓
Quality management		✓	✓	✓	✓	✓	✓	✓
Workflow management	✓	✓	✓	✓	✓	✓	✓	✓

3.6.1 Features and capabilities

The essential and distinguishing features and capabilities of the analyzed low-code platforms can be summarized as follows. *OutSystems* provides developers with a quick mechanism to publish developed applications, the capability to connect different services, to develop responsive mobile and web-apps, security mechanisms and real-time dashboards. *Mendix* supports collaborative project management and end-to-end development, pre-built templates with app stores and interactive application analytic. *Zoho Creator* has an easy-to-use form builder, user-friendly and mobile-friendly user interfaces, and the capability of app-integration among different Zoho CRM apps, Salesforce, etc. It also supports pre-built templates and customized workflows. *Microsoft PowerApp* supports integration with Microsoft Office 365, pre-built templates, easy mobile and tablet application conversion, and the capability to connect with third-party applications for basic application development. *Google App Maker* has a drag-and-drop function similar to most of the analyzed low-code platforms, app preview, reusable templates, deployment settings, means to specify access roles, built-in tutorials and google analytic integration. *Kissflow* supports progress tracking, custom and pre-built reports, collaborative features, and the possibility to use third-party services such as Google document, and Dropbox documents. It also supports Zapier to integrate different systems. *Salesforce App Cloud* has an extensive app market place for pre-built apps and components, reusable objects and elements, in-built kanban boards and a drag-and-drop process builder. *Appian* supports native mobile apps, drag-and-drop tools, collaborative task management, and a decision engine with AI-enabled complex logic.

3.6.2 Additional aspects for comparing LCDPs

The taxonomy discussed in the previous section plays an important role when users have to compare candidate LCDPs and select one among possible alternatives. Further than the features previously presented, we identified additional aspects that are orthogonal to the presented taxonomy, and that can be taken into account when decision-makers have to decide if a low-code development platform has to be adopted and which one.

Type of solutions to be developed: there are two main types of applications that can be developed employing LCDPs, namely B2B (Business to Business) and B2C (Business to Customer solution). B2B solutions provide users with business process management (BPM) functionalities such as creation, optimization, and automation of business process activities. Examples of B2B solutions include hotel management, inventory management, and human resource management. Multiple applications can be combined in a B2B solution. B2C solutions provide more straightforward answers for end customers. B2C solutions are for developing single applications such as websites and customer relations management applications. The interactivity aspects of B2C is much more crucial than B2B ones.

Size of the user's company/organization: another dimension to be considered when selecting LCDPs, is the size of the company/organization that is going to adopt the selected LCDP. Organizations fall under three possible categories: *small* (with less than 50 employees), *medium* (if the number of employees is in between 50 to 1000), *large* (if the number of employees is higher than 1000). Thus, the decision-maker must keep in mind the organization size to identify the optimal solution according to her needs. Any organization who wishes to scale their enterprise at an optimum cost need to select an LCDP based on the strength of the company. LCDPs such as Salesforce app cloud, Mendix, and OutSystems support large enterprises, and they are used to develop large and scalable applications. Google App Cloud, Appian, Zoho Creator are instead mainly for supporting small to medium scale enterprises and they are relatively cheaper.

Cost and time spent to learn the platform: the time spent on the development, testing and deployment of an application may vary from one low-code platform to another. To be proficient in such processes, users must spend time to learn all the related aspects of that platform. Also, decision-makers have to consider potential training costs that have to be faced for learning the concepts and processes of that particular low-code platform.

The price of the low-code platform: it is one of the most critical criteria, especially for small or medium-scale companies. The price of the platform can be estimated as the price of using the platform for one developer per month. Moreover, the dimensions that contribute to the definition of the price include *i)* the number of applications that need to be deployed, and *ii)* where data are going to be stored, i.e., in on-premise databases, in cloud environments, or in hybrid configurations.

Increase in productivity: The adoption possibilities of low-code development platforms have to be assessed by considering the potential number of developed applications with respect to the time spent to learn the platform, the price incurred in training and to buy the licenses to use the considered platform.

3.7 Experience Report

The making of such platforms capable of giving citizen developers the ability to build fully-fledged applications faster and efficiently comes on a cost. Critical architectural decisions are made to ensure minimal coding, speed, flexibility, less upfront investment and out-of-box functionalities that deliver the full application faster. However, decisions that are usually taken during the usage of LCDPs can give place to some issues that might emerge later on. In particular, to get insights into LCDPs, we developed the same benchmark application by employing different platforms. The benchmark application is a course management system intended to facilitate trainers and trainees to manage their courses, schedules, registrations and attendance. Despite the simplicity of the application, it exhibits general user requirements that are common during the development of typical functionalities such as management of data, their retrieval and visualization. Moreover, we had the possibility of integrating external services via third-party APIs,e.g., to show maps. We managed to investigate how reusable code and artefacts developed in one platform can be integrated into other low code platforms hence smoothing the path toward discovery and reuse of already proven artefacts across different platforms.

The first performed activity to develop the benchmark application was the elicitation of the related requirements. We came up with the corresponding use cases, and thus with the functional requirements of the system.

Low-code development platforms are suitable for organizations that have limited IT resources and budget because delivers the fully featured product in a short span. However it was noticed than the focus is on functionality of the modules, thus limiting the user to operate according to her own requirements. Third party integration is limited and the management and maintenance of the application is limited due to limited extensible capabilities to add new functionalities with custom code. Plus, it is tough to edit pre-built components to your own liking.

Below are what we notice as major challenges that transcend most of the low-code development platforms we surveyed. Users and developers are likely to face these challenges along the course of development in LCDPs such as interoperability issues among different low-code platforms, extensible limitations, steep learning curves, and scalability issues [78] [104][116] as discussed below.

Low-code platforms' interoperability: this characteristic ensures interaction and exchange of information and artefacts among different low-code platforms, e.g., to share architectural design, implementation or developed services. Unfortunately, most of low-code platforms are proprietary and closed sources. There is a lack of standards in this domain by hampering the development and collaboration among different engineers and developers. Thus, they are unable to learn from one another and the reuse of already defined architectural designs, artefacts and implementations is still hampered.

Extensibility: the ability to add new functionalities not offered by the considered platform is hard in such proprietary platforms or even impossible. Due to lack of standards, some of them require extensive coding to add new capabilities, which have to adhere to architectural and design constraints of the platform being extended.

Learning curve: most of the platforms have less intuitive graphical interfaces. For some of them, drag-and-drop capabilities are limited, and they do not provide enough teaching material, including sample applications and online tutorials to learn the platform. Consequently, the platform adoption can be affected. The adoption of some platforms still requires knowledge in software development, thus limiting their adoption from citizen developers who are supposed to be the main target of these platforms and products.

Scalability: Low code platforms should be preferably based on the cloud and should be able to handle intensive computations and to manage big data, which get produced at high velocity, variety, and volume [90]. However, due to lack of open standards for such platforms, it is very challenging to assess, research and contribute to the scalability of these platforms.

3.8 Summary

In recent years, there has been a significant increase in interest in LCDPs from both academia and industry. However, with the availability of hundreds of low-code platforms [25], understanding and comparing them can be a daunting task without an appropriate evaluation framework. This technical evaluation of low-code platforms has never been discussed before, making this chapter the first to aim at analyzing different features associated with different low-code platforms. We analyzed eight relevant low-code platforms in this chapter to identify their similarities and differences and defined an organized set of distinguishing features to compare them. Additionally, we presented a short experience report highlighting the limitations and challenges we encountered while developing a simple benchmark application using the considered platforms.

As a result, we conducted an analysis of various low-code development platforms using a standard business process tool such as BPMN to understand their complete business process and data handling mechanisms. The next chapter will explain the data-handling capabilities used in different LCDPs and discuss the process modeling capabilities supported by each of them.

Chapter 4

Analyzing business process management capabilities of LCDPs

LCDPs offer workflow modeling notations to define business processes and automate the corresponding tasks, including data change, scheduling, or connecting information flows to external services. For example, when an employee is added to a company, an automated welcome email is sent to give some essential information. Such automation helps avoid manual and repetitive work and can produce a better and more efficient result in the overall business process of that company.

The use of LCDPs to automate business processes aims at reducing the cost and time of implementing and maintaining software systems by satisfying different groups of technical and non-technical users, e.g., software developers and business analysts, respectively [120]. Furthermore, such business process automation helps develop various applications such as event monitoring, process automation, approval process control, escalation management, inventory management, quality management, and workflow management. [97]. As discussed in [51], business process management aims to support three main activities: *i)* modeling business processes to capture different workflow specifications, *ii)* optimizing business processes, and *iii)* automate the workflow to generate its implementation. LCDPs provide modeling constructs to specify workflows and handle corresponding data in this respect. One of the critical use cases of LCDPs is to automate business processes, workflows, and case management that involves complex business logic that can operate with external services and multiple end-user roles [117]. However, selecting the right LCDP that can be employed to specify, automate, and manage the business processes at hand is a difficult task, which is further complicated by the increasing number of LCDPs being continuously released.

In this chapter, we analyze and compare eight low-code development platforms by focusing on their capabilities for specifying and managing business processes. The considered definition of business processes is based on the standard Business Process Modeling and Notations (BPMN) [6]. BPMN plays the role of the reference modeling language to support the comparison and the analysis of the considered LCDPs.

This chapter elaborates on the following research activities.

- We analyze the workflow and data handling mechanisms of different LCDPs associated with their respective process modeling constructs.
- We identify the existing and missing mapping between components in BPMN-like modeling language and the analyzed low-code platforms.
- We discuss a set of quality criteria inspired by BPMN that can be used to evaluate modeling constructs of different LCDPs.

4.1 Process modeling with BPMN

Process modeling is widely used within organizations as an approach to describe through graphical notations how business entities conduct their operations [87] in terms of activities, events, and control flows. In addition, process models may also represent information about involved data, resources, and other related artifacts. Petri nets [88] and BPMN[1] are two prominent examples of process modeling techniques. BPMN was developed by Object Management Group[2] (OMG) and was released for the first time in May 2004. The BPMN specification was created to provide a notation intended to be *"understandable by all business users, from the business analysts who create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and, finally, to the business people who will manage and monitor those processes"* [121].

In BPMN, business processes are generally called workflows, and they are specified in terms of a dedicated notation consisting of various modeling constructs for activities, events, gateways, etc. In addition, BPMN also includes composition mechanisms, e.g., orchestrations, endowed with corresponding execution semantics. The available constructs help technical and non-technical individuals to understand their views of the business model by hiding or revealing the necessary detail according to the role, designation, or expertise of the person involved. The execution of BPMN specifications is typically performed by transforming models to an XML-based business process execution language (BPEL) that helps to optimize and automate the defined business processes [81]. BPMN 2.0 [6], released in 2011, expands earlier BPMN 1.x capabilities adding choreography diagrams for modeling several business interactions. Overall, BPMN consists of notations and semantics of three different diagrams, i.e., *collaboration*, *process*, and *choreography* diagrams. In this paper, we focus on the core BPMN elements for modeling business processes as shown in Table 4.1.[3]

TABLE 4.1: BPMN core elements

Core Elements	**Short Description**
Swimlanes	
Pool	Defines a process participants or an external process. Can have multiple lanes.
Lane	Define a specific participant or role within a process inside a pool.
Flow Object	
Event	Shows something that has happened or may about the happen during a process.
Activity	Work performed within a process by a participant.
Gateway	Controls the sequence & flow within a process, providing routing within a process.
Data	
Data Object	Information required or produced by an Activity that persists only during a process.
Data Store	Information required or produced by an Activity that persists beyond the process.
Message	Contains the information between two participants (usually across pools).
Artifacts	
Group	Allow flow objects to be grouped for documentation and analysis.
Annotation	Provides additional information on an element within a process.
Connecting Objects	
Sequence Flow	The connection between flow object in a process shows execution within a process.
Message Flow	Shows message between process participants (usually across pools).
Association	Links Data/Artifacts to flow object shows the direction.

In the following, we describe an illustrative and running example of a *Recruitment Drive* process. The applicant reads the job notification on the company's website and checks her eligibility criteria. If ineligible, she will close the website; otherwise, she clicks on the application link given on

[1] https://www.omg.org/spec/BPMN/2.0/
[2] https://www.omg.org/
[3] Some of the considered BPMN core elements are taken from https://www.omg.org/bpmn/Samples/Elements/Core_BPMN_Elements.htm

the notification. She further fills out the application and submits it. This will notify the Human Resource (HR) manager, who checks and validates the application. If the application is incorrect, the application will be rejected, and thus the application form gets closed. Alternatively, if the application is correct, an interview is set up. Finally, the manager sends an email to the applicant notifying the time and place for the interview. Consequently, the manager also conducts interviews in which the applicant participates. After the interview, the manager sends the pass or fail message to the applicant. If the applicant passes the interview, the manager sends the applicant's pass message to the administrative department. Upon receiving the notice from HR, the administrative department issues a registration sub-process to complete the applicant's hiring.

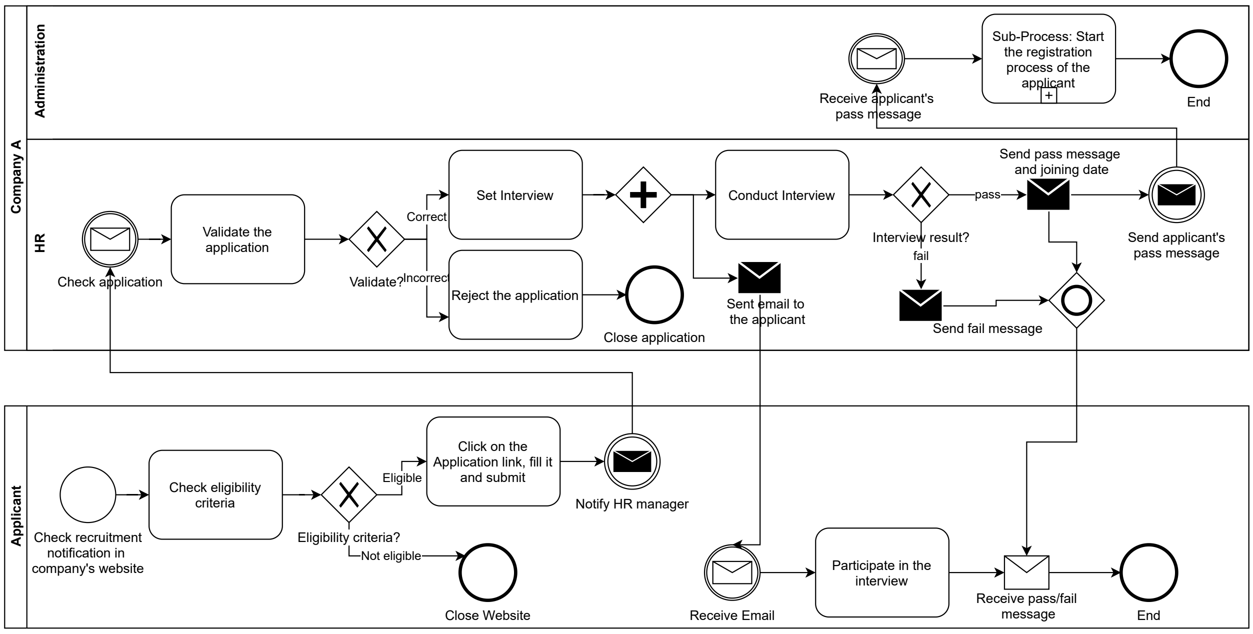

Company A
Administration
Receive applicant's pass message
Sub-Process: Start the registration process of the applicant
End
HR
Check application
Validate the application
Validate?
Correct
Incorrect
Set Interview
Reject the application
Close application
Conduct Interview
Sent email to the applicant
Interview result?
pass
fail
Send pass message and joining date
Send fail message
Send applicant's pass message
Applicant
Check recruitment notification in company's website
Check eligibility criteria
Eligibility criteria?
Eligible
Not eligible
Click on the Application link, fill it and submit
Notify HR manager
Close Website
Receive Email
Participate in the interview
Receive pass/fail message
End

FIGURE 4.1: Fragment of the BPMN2.0 specification of the *Recruitment Drive* example

Figure 4.1 depicts a fragment of the BPMN2.0 specification of the *Recruitment Drive* example, which has been modeled employing core elements shown in Table 4.1. There are two *lanes* namely `Company A` and `Applicant`. The former contains two *pools* namely `HR` and `Administration`. The applicant sends the application *message* to notify `HR`. The `HR` sends back the interview intimation to the applicant followed by pass or fail *message* of the interview based on the fail decision making *gateway*. Also, `HR` sends a pass *message* to the admin based on the pass decision making *gateway* and the admin starts the registration sub-process *activity*. The *flow object* from one *event* to an *activity* is also shown multiple times in Fig. 4.1. For instance, the start *event* "Check recruitment notification from company's website" to the *activity* "Check eligibility criteria" is attached via a *flow object*. Also, there are multiple *message flow* elements connecting different send and receive *message*s that run across *pools* or *lanes*.

4.2 Analysis of process modelling languages

This section discusses works that analyze process modeling languages and compare them based on relevant criteria. The purpose of this section is to give a mini systematic literature review by following consolidated guidelines for this kind of works in the area of software engineering [69]. To identify and discuss current results, we observed a process consisting of the following steps: *(i)* definition and execution of a query string to automatically collect papers of interest based on a set of keywords; *(ii)* manual filtering the retrieved papers with respect to defined inclusion and exclusion criteria; *(iii)* discussion of the remaining papers by answering the following research question:

> **RQ:** What are the key aspects that are considered when comparing business process modeling languages?

Details of the applied methodology are given in Sect. 4.2.1. The obtained results are presented in Sect. 4.2.2, whereas the answer to the defined research question is given in Sect. 4.2.3.

4.2.1 Study Design

As previously mentioned, we defined a query string to be executed on different digital resources to collect papers that analyze existing work comparing business process modeling languages. To this end, the *query string* that we conceived is the following:

> ("BPMN" OR Business "Process Modeling Language") AND (("Qualitative Comparison" OR "Analysis") AND ("Business Process Modeling Languages" OR "BPMN"))

It has been applied on the following *electronic resources*:

1. *IEEE Xplore*[4];
2. *Science Direct*[5];
3. *Association for Computing Machinery (ACM) Digital Library*[6];
4. *Springer Link*[7];
5. *Academia*[8];
6. *Scientific Research*[9].

We decided the selection criteria of this study before the query definition phase to reduce the likelihood of bias. In the following, we describe the inclusion (I) and exclusion (E) criteria of our study:

- I1 Papers that have been published in the period 2000–2022;
- I2 Studies subject to peer review (e.g., papers published as part of conference proceedings or journal publications are considered, whereas white papers are discarded);
- I3 Studies that qualitatively compare different modeling languages and are available for consultation;
- E1 Papers that are not written in English;
- E2 Short papers of tutorial slides;
- E3 Book chapters.

Applying such inclusion and exclusion criteria to the papers initially collected by the query execution, we obtained 10 documents that are discussed in the next section.

[4]https://ieeexplore.ieee.org/Xplore/home.jsp
[5]https://www.sciencedirect.com/
[6]https://dl.acm.org/
[7]https://link.springer.com/chapter/10.1007/978-3-319-31232-3_58
[8]https://www.academia.edu/
[9]https://www.scirp.org/journal/index.aspx

4.2.2 Results

In Bendraou et al.(2010) [24], the authors discuss the Software Process Modeling Language (SPML) and compare it with the UML-based SPML. The paper focuses on the framework that involves setting up the requirements for process modeling: semantic richness (expressiveness), modularity, executability, conformity to UML-standard, and formality. This paper also evaluates the suitability of the known UML-based SPMLs for process modeling.

The paper co-authored by Bendraou et.al. [24] focuses on the criteria that distinguish different SPMLs like semantic richness and expressiveness. These aspects include core process elements such as activity, artifact, role, and agent. One of the other aspects of expressiveness is activities, and action coordination that involves proactive control and reactive control [122]. Proactive controls specify the order in which the activities are executed. They are precedence relationships (i.e., start-start, start-finish, finish-start, and finish-finish) and call actions such as explicitly calling an activity, operation, etc. Following proactive control, reactive control specifies conditions or events in response to the executed activities. Reactive control includes Exceptions, Event handling, etc. Also, exception handling is one of the criteria for expressiveness that explains the recoverability of a stable state of the process that handles failure.

Garcia et al. [49] describe a Quality Model (QM) used in a framework called Quality Evaluation Framework (QuEF) that manages the quality requirements of a modeling environment. The quality model formulates different approaches that characterize the quality of model-based SPML. These approaches include expressiveness, understandability, conformity to standards, granularity, executability and orchestability, measurability, business rules, supporting tools, and validation in real environments. With the help of these approaches, ten representative SPMLs are evaluated and compared. These SPMLs are of three types: metamodel-level approaches, SPML based on UML, and methods based on standards. Lastly, this paper has many model-based proposals for SPML. Furthermore, this paper has taken newer aspects to characterize SPMLs. Such characteristics are validation within enterprise environments, friendly support tools, mechanisms to carry out continuous improvements, mechanisms to establish business rules, and elements for software process orchestration. All the quality criteria mentioned in this paper have been described in Section 4.5 concerning various LCDPs and compared with the BPMN modeling standard.

In Portela et al.(2012) [85] the authors analyze how SPEM (Software Process Engineering Metamodel Specification) and BPMN (Business Process Modeling Notation) standards are represented in a software process modeling context. This analysis is done by defining a standard structure based on a process ontology to define a software process. Then the SPEM and BPMN standards are semantically matched with the default modeling process. These notations are also mapped with components of quality models named CMMI-DEV (Capability Maturity Model Integration for Development) and MR-MPS (Reference Model for Software Process Improvement). This analysis of the above-used standards provides best practices through specific characteristics necessary to model and represent the software process. As we have not used the quality models such as CMMI-DEV or MR-MPS in the paper, the modeling constructs of various LCDPs are not semantically compared in a standard way. However, an LCDP can easily interpret any core elements of BPMN.

In paper [85], various standard process structures with modeling elements are considered. These modeling elements are Process, LifecycleModel, Combination, Activity, Artifact, Resource, Procedure, PatternofActivity, and Restriction. With these software structural (modeling) elements, SPEM and BPMN notations correspond to the two quality models (CMMI-DEV and MR-MPS). Further representativeness of SPEM and BPMN notations are analyzed by considering characteristics such as Expressiveness, Reuse, Management, Evolution, Multilevel, Understanding, and Organizational Integration. Finally, various modeling elements of BPMN are discussed and matched with the low-code platform. These matching are based on quality criteria such as expressiveness, understandability, conformity to standards, granularity, executability and orchestability [49].

In Guizani et al.(2021)[54], the authors select a business process modeling language by analyzing the multi-criteria depending on the modeler's requirements. These criteria are expressiveness, flexibility, formality, readability, support tools, usability, and ease of learning. The mentioned criteria help to evaluate different business process modeling languages such as BPMN, Event-driven Process Chain (EPC), Petri Net, UML-Activity Diagram (UML-AD) and Yet Another Workflow Language (YAWL). The modeler can provide different weights to the criteria based on the modeler's requirements which will be calculated to select the most appropriate modeling language. Similarly, paper [55] also uses an extensive multi-criteria evaluation method to choose a sensitive business process modeling to improve the localization, identification, and characterization of the most critical knowledge. Our current work does not provide any weights to the quality criteria of the modeling construct of BPMN and that of the various LCDPs. However, the modeler can give the weights according to her requirements in developing a particular application using a specific LCDP. For example, a developer may give more importance to the expressiveness of the modeling construct and less to the readability or the ease of learning criteria. On the other hand, a naive non-developer can give more importance to the readability and ease of learning and less to the expressiveness of a modeling construct.

Wang et al. [119] discusses some major business process modeling standards by considering quality criteria such as metamodel, graphical notation, serial representation, extensibility, and tool support. The considered business process modeling approaches in this paper are BPEL4WS (Business Process Execution Language for Web Services) [11], BPMN (Business Process Modeling Notation), UML (Unified Modeling Language) [47], XPDL (XML Process Definition Language) [123], Petri Net [83] and IDEFO (Integration Definition Method) and IDEF3 [39].

In paper Pereira et al.(2016) [82], the strength and weaknesses of the several business process modeling languages are discussed so that a comparative perspective can be drawn among them. The comparative framework is based on relevant criteria such as expressiveness, readability, usability, user friendly, formality, versatility, universality, tools support, and flexibility. These criteria are compared among the five process modeling languages such as BPMN, Event-driven Process Chain (EPC), UML-Activity Diagram (UML-AD), Role Activity Diagram(RAD) and Integration Definition (IDEF). The discussed framework quantifies the level of support that each process modeling language is chosen based on the values of each criterion.

In Kelemen et al.(2013) [59], the selection of a process modeling is characterized by several quality criteria, such as intelligibility, the ability to express process elements and workflow patterns along with the software support, and portability. Furthermore, the mentioned quality criteria characterize the modeling languages such as BPMN, EPC, and UML. A comparative analysis is done to unify models and standards by creating a unified process that could be useful in other projects in the industry.

In Birkmeier et al.(2010) [28], authors evaluate the usability of BPMN and UML-Activity Diagram (UML-AD) for many business users based on an empirical comparison to justify their ability to express and communicate business process semantics. Furthermore, the paper claims that process modeling through UML-AD is similar to that of BPMN in terms of user effectiveness, efficiency, and user satisfaction in a specific context of architecture development scenarios.

4.2.3 Key aspects for comparing business process modeling languages

By referring to Table 4.2, this section addresses the research question underpinning the performed study. In particular, we analyze the related works that have been summarized in the previous

TABLE 4.2: Quality criteria to compare process modeling approaches in the literature

Quality criteria	Existing studies
Conciseness	[24],[49],[54],[55],[119],[82],[59],[28]
Executability	[81], [24], [49], [119]
Standardization	[24], [49], [85], [119]
Granularity	[24], [49], [85], [54], [119]
Collaborativeness	[49], [85]

section with respect to different qualitative characteristics. According to the performed analysis, the broader criteria to differentiate process modeling languages are *conciseness*, *executability*, *standardization of process models* such as UML or BPMN, *granularity* and *collaborativeness*.

Conciseness Two kinds of properties focus on conciseness. They are *semantic* and *syntactic* properties. Semantic properties enable the richness in explaining the different states of process models that can be replicated in advanced complex real scenarios and reducing the redundant features to showcase process models. In contrast, the syntactic properties focus on the user-friendliness of the process modeling tool. Papers such as Bendraou et.al. [24], Garcia et al. [49], Portela et al. [85], Guizani et al. [54], Hassen et al. [55], Pereira et al. [82], Kelemen et al. [59] and Birkmeier et al. [28] show key importance in the expressiveness and the understandability of the process modeling language. They also studied the modularization, reusability and universality of the modeling language.

Executability The execution of the business process models is integral to developing a process-oriented application. For example, an LCDP focuses on executing the process models using data models and logic applied in creating an application. Also, the execution of BPMN specifications is done by transforming models to an XML-based business process execution language (BPEL) that helps to optimize and automate the defined business processes [81]. But the use of BPEL is only sometimes used nowadays. Executability criteria are considered in papers such as Bendraou et.al. [24] and Garcia et al. [49].

Standardization of process models The standardization and formalization of process models showcase the clarity in defining particular process model elements, which can be used for large groups of people. There are papers such as Bendraou et.al. [24], Garcia et al. [49] and Portela et al. [85] that consider standardization as one of the criteria to evaluate different process models.

Granularity A process model can be subdivided into logical and syntactical forms based on user demands. This granularity can be either coarse-grained or fine-grained based on the complexity and the user's requirement to build a particular process model. For example, papers such as Bendraou et.al. [24], Garcia et al. [49], Portela et. al. [85] and Guizani et al. [54] use granularity to characterize various process modeling language.

Collaborativeness Multiple people can work to build on the same process model while maintaining the cohesiveness of the work done, especially from multiple remote locations. For example, papers such as Garcia et al. [49], and Portela et al. [85] allow collaboration to validate and build process models in real environments.

To summarize on comparing modeling constructs of different process modeling tools, we have used different quality standards defined in the paper [49]. Gartner forecasts that market for LCDP would grow by 23% in 2022-2023 [50]. To assist this growth, it is important to understand the data-handling and process modeling constructs used in different LCDPs. Therefore, the study to compare LCDP and BPMN focuses on using data-handling capabilities in their process modeling framework that are executable to perform various application tasks.

The performed analysis of the related work shows various methods in which several defined quality standards are formulated to define a framework that can compare multiple modeling constructs. No quantitative analysis has been done to compare different process modeling tools. It is worth noting that previous works have yet to consider modeling constructs of any LCDP and compared them with other modeling technologies.

4.3 Discussing LCDPs process modelling and data handling constructs

According to a Gartner report [117], leading low-code platforms are Outsystems, Mendix, Microsoft Powerapps, and Salesforce Lightning. In this section, we consider all of them in addition to the recent and niche [117] platforms such as Amazon Honeycode, Google Appsheet, Zoho Creator, and Thinkwise. Most of these platforms use similar concepts in graphical user interfaces, allowing developers to define and manipulate data specified through tables, forms, reports, and other kinds of representation. LCDPs permit importing data tables from external databases or creating new ones using the platform's data models. Data models or imported data tables can be edited, which then makes a `form` screen where the application user gives input to it. The input data is then shown in a `report` screen, which can still be edited/updated by the user. Lastly, the input can be manipulated to give output in charts, graphs, etc. The `form` and the `report` can be edited or managed using UI controls such as link icons, buttons, etc. These generalized basic steps in using LCDPs to develop applications can be further extended by using different platform-dependent facilities to manage interoperability and collaboration [97].

This section discusses different LCDPs by referring to the previously presented "Recruitment Drive" example. For the sake of presentation and explainability, we focus on modeling explanatory parts of the running example. In particular, we focus on the following steps: on clicking the user's application link, the detail and the user's eligibility information are stored in the database. If the eligibility information given by the user equates with the considered eligibility criteria of a post set by the company, then the recruitment system application would send a pass message to the user; otherwise, the user will get a fail message.

The considered scenario is assumed to rely on two data tables, namely `Applicant` and `Recruitment Notification`. Therefore, the implementation of the case study with the LCDP at hand starts with the creation of two tables. An `Applicant` table consisting of the fields *Name of the applicant*, *Email Id*, *Eligibility Criteria* and *Post Applied*. A `Recruitment Notification` table has to be also created with the fields *Name of the Post* and *Eligibility Criteria*. The table `Applicant` is linked to the `Recruitment Notification` through the foreign key *Post Applied*. This also means that the *Post Applied* of the `Applicant` table is a lookup variable for the *Name of the post* field inside the `Recruitment Notification` table. Each applicant can apply for one or more posts in the `Recruitment Notification` table. A user on a post fills an application form which is then stored in the `Applicant` table. Suppose the *Eligibility information* of the applicant matches with the *Eligibility criteria* mentioned in a specific *Post name* row in the `Recruitment Notification` table. In that case, a notification email/message is sent to HR/applicant mentioning the applicant success report. If the *Eligibility information* of the applicant does not match the *Eligibility criteria* then the whole row of that particular applicant is deleted from the `Applicant` table. Other minor details of the case study workflow are not considered as it would be repetitive to model similar business process elements with similar logic.

As anticipated, in the following sections, we discuss the specification of the described case study given with different LCDPs. It is worth noting that we are comparing BPMN modeling elements to build the case study application (shown in Fig. 4.1) with its equivalent modeling constructs using different LCDPs associated with the data handling mechanism and their implementation. The analysis is presented from three different points of views, i.e., the *core* modeling elements of BPMN which is then compared to the implementable *workflow* of the eight considered LCDPs, in addition to their specific *data* handling mechanisms.

4.3.1 OutSystems

OutSystems [79] is a low-code development platform offering a visual, model-driven development environment with industry-leading AI-based assistance. The development environment provided by Outsystems is a suite consisting of Service studio and Integration Studio.

Core Modeling Elements: `OutSystems` supports BPMN `Event` by means of different modeling elements including *Start*, *Conditional Start*, *End* and *Wait*. `Activity` is referred to as *Human Activity*, *Automatic Activity*, and *Execute Process*. Also, `Gateway`, `Message` and `Annotation` are supported by means of the modeling constructs *Decision*, *Send Email*, and *Comment*, respectively.

Workflow Modeling: Workflows defined in OutSystems typically handle different business processes such as invoices, processing orders, or handling complaints, also known as BPT (Business Process Technology) [79]. The key feature in developing business processes in OutSystems is the separation of the process development built by *steps*, *interfaces*, *logical structure*, and *databases*. Steps define the flow of the process under development. Interfaces enable different ways to access the application. For instance, interfaces enable the parts of the app to be shown in the UI, e.g., charts, forms, etc. Logical structures build reusable logic blocks and reusable integration components like SOAP or REST services. Databases are used to manage and store data, even from external sources.

Data handling capabilities: In the Data section of Outsystems' user interface Service Studio, a table can be created by adding an entity to the in-built cloud database or importing a table from Excel. When the table is imported from Excel, it can be transformed into the user-defined editable table in the platform if needed. Moreover, querying and fetching individual data from an Excel sheet is not possible directly from/to the external data without using an external API. A developer manipulates data or attributes of the table only after importing the table to the OutSystems environment. Integration of other external databases is done in `Integration Studio`[10] by configuring the extension to use a database connection and then refer and use it. Such integration is an interoperability feature of the platform and would require API integration with some coding; therefore, these integration can be challenging for inexperienced developers.

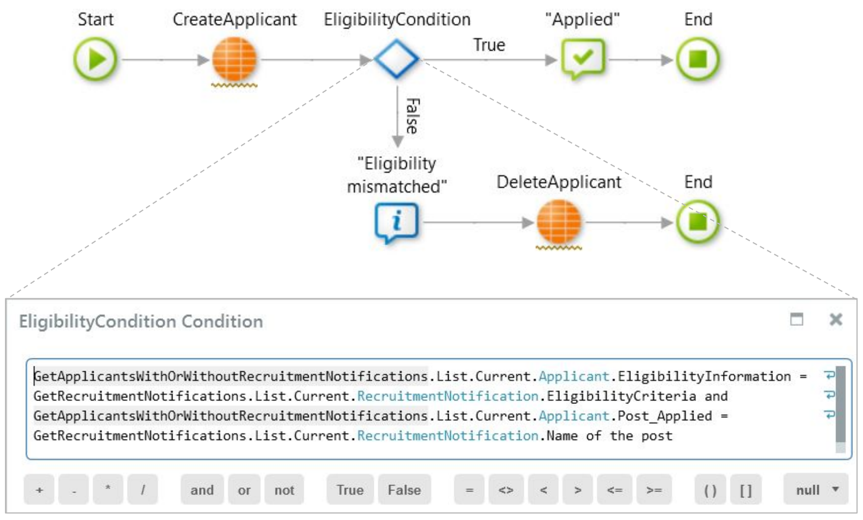
Figure 4.2 shows the OutSystems user interface Service Studio at work, which is the development environment that users can use to model business applications [70]. Fig. 4.2 shows the specification of the applicant eligibility explained above with a dedicated scripting language. The reported script assures that if the stated eligibility condition is matched, the applicant and the HR manager receive a message that the applicant successfully applied for the post. On the other hand, if the eligibility condition does not match, the current row in the "Applicant" table is deleted, meaning the applicant cannot apply for the chosen post.

4.3.2 Mendix

Mendix[11] offers both visual-modelling and low code integration functionalities. It uses a dedicated modeling tool called Microflows to express the application's logic.

Core Modeling Elements: `Mendix` has their `Event` element termed as *Events* only and `Activity` is termed as *Activities* and *Error handlers*. Also, `Gateway`, `Data Objects`, `Sequence Flow` and `Association` are termed as *Splits*, *Input parameters*, *Flows*, and *Connectors*, respectively.

[10] `https://success.outsystems.com/Documentation/11/Reference/Integration_Studio`
[11] `https://www.mendix.com/`



```
GetApplicantsWithOrWithoutRecruitmentNotifications.List.Current.Applicant.EligibilityInformation =
GetRecruitmentNotifications.List.Current.RecruitmentNotification.EligibilityCriteria and
GetApplicantsWithOrWithoutRecruitmentNotifications.List.Current.Applicant.Post_Applied =
GetRecruitmentNotifications.List.Current.RecruitmentNotification.Name of the post
```

FIGURE 4.2: OutSystems business flow

Workflow Modeling: The process of creating a new application in Mendix starts with the creation of the data model. Then, the user needs to edit the fields of the entity/table of the app. This is followed by creating an object linked to an entity that needs to be managed to add/update records. Once these steps are completed, the user can build processes employing the available Microflow modeling language. Microflow is based on the BPMN standard, and it helps to extend or change the default behavior of the developed application [72]. Microflow also defines several business processes and integrates with external systems, databases, or web services. Microflows decide the flow of the defined pages involved in creating the logic for the application. This would extend or create new business processes and show the application's extensibility by integrating external services and databases.

Data handling capabilities: The table is created by designing a domain model inside the Mendix platform. In this domain model, the user can define all the attributes and relationships among the specified table. Although it is possible to connect it to an external database, it requires separate import modules to connect to different data sources. However, it is not easy to use such import modules for non-expert developers.

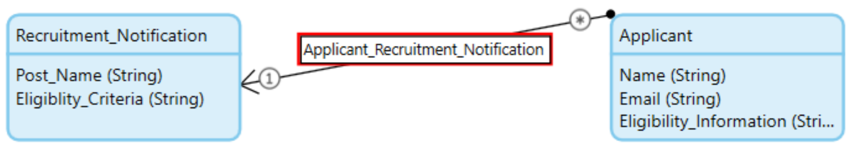

Recruitment_Notification
Post_Name (String)
Eligiblity_Criteria (String)
Applicant_Recruitment_Notification
1
*
Applicant
Name (String)
Email (String)
Eligibility_Information (Stri...

FIGURE 4.3: Recruitment Drive Domain Model in Mendix

Figure 4.3 and Fig. 4.4 show fragments of the data model and of the workflow specification given in Mendix, respectively, of the explanatory example. According to the data entities defined in Fig. 4.3, an applicant can apply for exactly one post at a time, as can be noticed from the target cardinal of the association from the entity `Applicant` to `Recruitment_Notification`. The defined data entities and their relationships underpin the specification of corresponding form pages enabling the execution of CRUD (Create, Read, Update, Delete) operations. For instance, the form related to the `Applicant` entity refers to the values of the attribute *Post_Name* that can be retrieved by a drop-down variable attained from the `Recruitment_Notification` table.

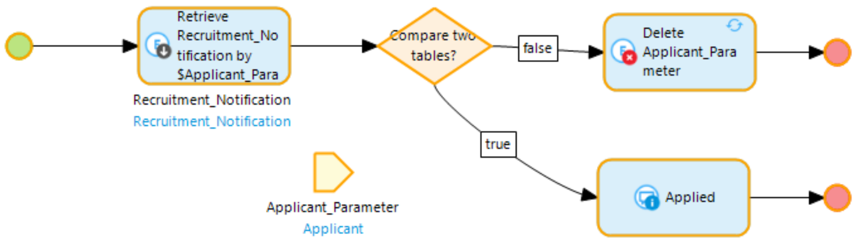


FIGURE 4.4: Recruitment Drive Microflow in Mendix

Microflow allows specifying workflows as shown in Fig. 4.4. In the example, the entity `Applicant` of a particular row is parameterized, and the associated data linked to a specific row in the `Recruitment_Notification` entity is retrieved. Then, the decision flow is used to compare the entered eligibility information in the *Applicant* form with the eligibility criteria of the *Recruitment_Notification* form. If they match, a message is shown to inform the applicant that the application has been successfully applied. Otherwise, the current row in the *Applicant* list is deleted.

4.3.3 Zoho Creator

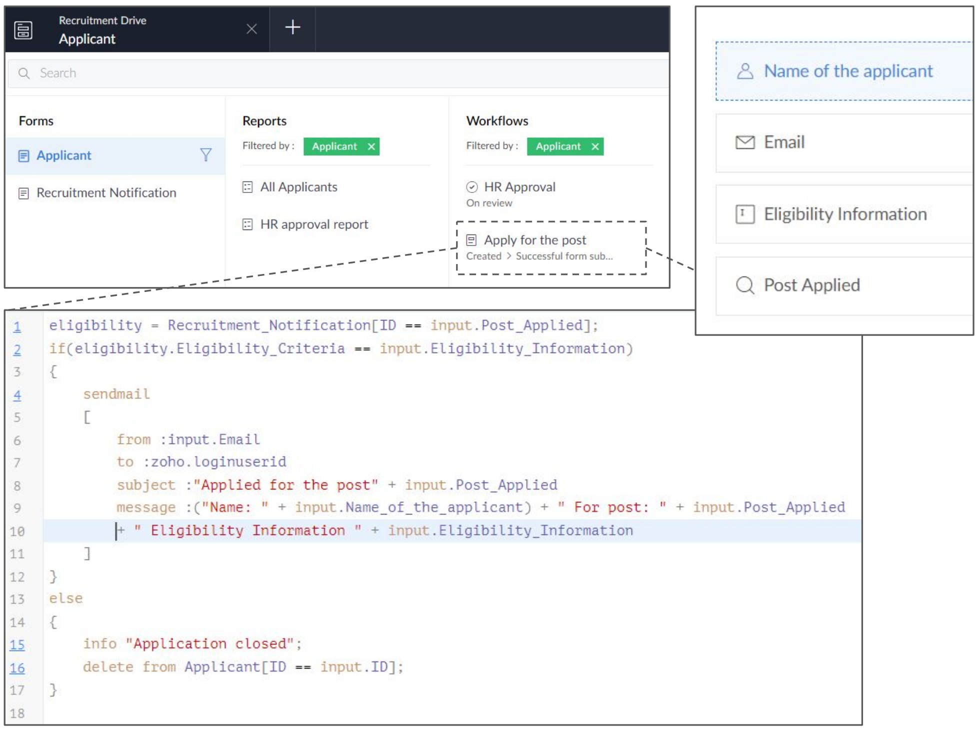
Zoho creator[12] is a cloud based software to create applications without coding experience or IT expertise. In Zoho Creator terminology, a data table is called *form*, whereas the view of a table is called *report*. The workflow in Zoho is created using a dedicated scripting language called Deluge [126].

Core Modeling Elements: In `Zoho Creator`, there are multiple elements matching with BPMN `Event`. In particular, the first `Event` setting, e.g., *"On a form event"*, *"In a schedule data"*, *"On an approval activity"*, *"On a payment activity"*, and *"On a function call"*, can be set by associating the action with the selected event. The second `Event` setting is called *"Record Event"*, that are sub-divided as *"Create"*, *"Edited"*, *"Created or Edited"*, and *"Deleted"*. The third `Event` setting is *"Form Event"* which is further divided as *"Before form submission"*, *"On form submission"*, and *"After form submission"*, referring to the form submission events. Likewise, `Activity` are broadly sub-divided into seven parts. They are *"Data Access"*, *"Notification"*, *"Field Actions"*, *"Integrations"*, *"Client Function code"*, *"Custom Action"*, and other *"Deluge scripts"*[13]. `Gateway` is implemented in terms of conditional statements provided by Deluge scripting language. Finally, ‘`‘Data Store’`’ is cloud-based and can be edited using Data Access code available in Deluge script that perform actions such as add record, fetch records, aggregate records, update records, for-each record and delete records.

Workflow Modeling: In Zoho Creator, rules and workflows can be defined by writing custom Deluge code to determine the behavior of an element of an application's page. The behavior construct can be conditional (if, else if, else, etc.), data access (add, fetch, aggregate, etc.), or client functions (hide/show, enable/disable, etc.) types. Workflow actions can be triggered by adding, updating, validating, or deleting page elements. Loading of forms can also be a possible workflow trigger. Also, button clicks and user inputs can trigger the execution of scripts building up the application. In Zoho Creator, there are three types of workflows: *i)* stand-alone scripts are used as a function to perform a task; *ii)* time-based scripts are used to schedule a task; *iii)* component-based scripts combine the first two workflow types that perform necessary actions on the forms/tables of interest.

Data handling capabilities: A table can be created by developing the form in the Zoho Creator environment, which is used to get the user input and automatically generate a list/report page.

[12]`https://www.zoho.com/`
[13]`https://www.zoho.com/deluge/`



```
eligibility = Recruitment_Notification[ID == input.Post_Applied];
if(eligibility.Eligibility_Criteria == input.Eligibility_Information)
{
    sendmail
    [
        from :input.Email
        to :zoho.loginuserid
        subject :"Applied for the post" + input.Post_Applied
        message :("Name: " + input.Name_of_the_applicant) + " For post: " + input.Post_Applied
        + " Eligibility Information " + input.Eligibility_Information
    ]
}
else
{
    info "Application closed";
    delete from Applicant[ID == input.ID];
}
```

FIGURE 4.5: Recruitment Drive in Zoho Creator

Alternatively, a developer can import an Excel or CSV file transformed into the platform's environment according to the developer's need to manipulate the required fields or data if needed. No direct manipulation of the external file is possible in such importing of data.

Based on the implementation description of the case study recruitment drive, two forms (i.e., tables) are created, namely *Applicant* (shown for convenience on the right-hand side of Fig. 4.5) and *Recruitment Notification*.

As specified in the script reported at the bottom of Fig. 4.5 the value of `Post_Applied` is retrieved from the *Recruitment Notification* form. The eligibility information in the `Applicant` table is equated with the eligibility criteria field in the *Recruitment Notification* form. If they match, then an email is sent to HR mentioning the detail for the *Applicant* form. If they do not match, then the whole current input row of the `Applicant` table is deleted.

4.3.4 Microsoft PowerApps

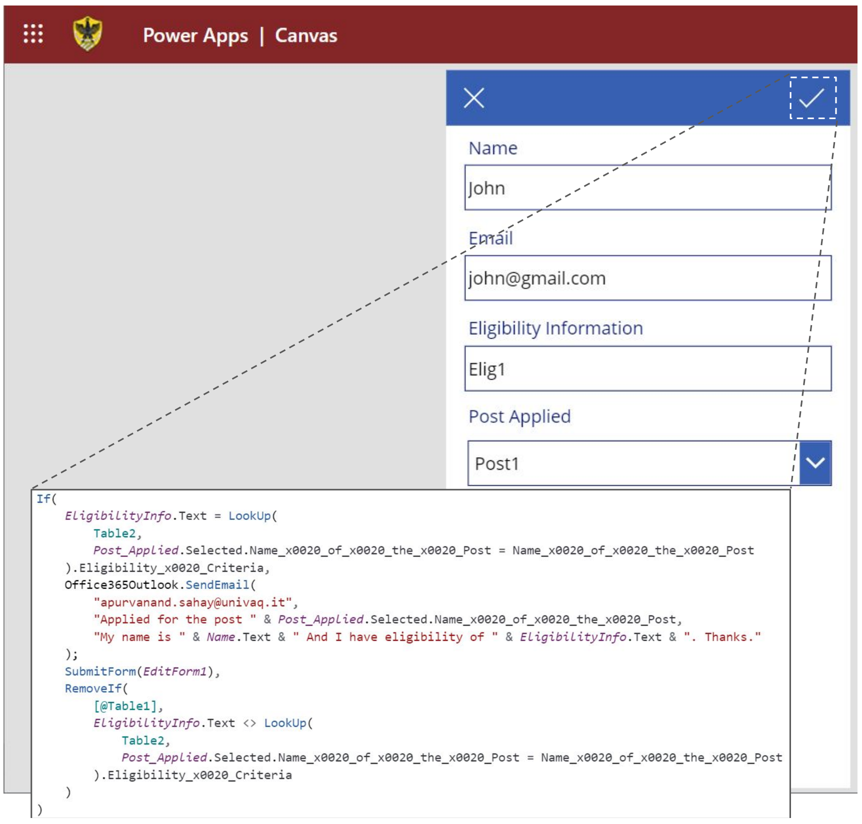
Also Microsoft has its own low code platform, and it is called PowerApps[14]. In Microsoft PowerApps, the workflow can be managed by using an OCL-like (Object Constraint Language) language [30] that can be applied to different objects of the modeled screens/pages as described below.

Core Modeling Elements: In Microsoft PowerApps, the BPMN `Event` element has the equivalent *Phase* construct, whereas `Activity` instances are developed with OCL (to manipulate or perform an action inside a given page), or in terms of *Workflow*, *Action Step* and *Flow step* elements to perform sequential activities. Also, the BPMN `Gateway` element is equivalently executed by the conditional element *if* and *switch* statement written in OCL code. Also, multiple connected data objects are used to connect different `Data Stores` such as MS SQL, Sharepoint, OneNote, OneDrive, etc. that correspond to the `Data Store` element in BPMN.

[14]https://powerapps.microsoft.com/

Workflow Modeling: In Microsoft PowerApps, there are three ways to model a workflow when building an application: *i)* the user can use multiple screens to develop complex forms consisting of different parts and pages having different formulae and controls; *ii)* the user can use a single page with complex OCL (Object Constraint Language) code; *iii)* a hybrid approach employs a single page, which integrates the business logic from external services such as MS Power Automate, Zapier, etc. In particular, MS Power Automate is the tool integrated into MS PowerApps for profiling business process steps and link them in the formulae of the defined fields or buttons according to the modeled logic [74]. MS Power Automate performs five types of workflows that can be incorporated into the PowerApps platform. They are *automated flow*, *instant flow*, *scheduled flow*, *user-interface flow*, and *business process flow*. Automated flows get executed once the corresponding trigger occurs, whereas instant flows will be executed based on the defined user's actions. Scheduled and user-interface flows work at a time interval or by clicking dedicated buttons on the application user interface. Finally, business process flows work on the parallel or sequential event-based actions defined by the developer.

Data handling capabilities: The connection to external databases (such as Microsoft Sharepoint, Dynamic 365, OneDrive, Salesforce, etc.) is one of the possible options offered by the PowerApps platform. Such a connection creates a common data model. Therefore, the developer can change the table's metadata (attributes, relationships, and data types). The user can fetch and manipulate data directly to and from the common data source. Those external data sources are linked with the PowerApps platform that requires authentication in all the associated services or databases. Custom-built tables can be exported to external databases if needed.



Name

John

Email

john@gmail.com

Eligibility Information

Elig1

Post Applied

Post1

```
If(
    EligibilityInfo.Text = LookUp(
        Table2,
        Post_Applied.Selected.Name_x0020_of_x0020_the_x0020_Post = Name_x0020_of_x0020_the_x0020_Post
    ).Eligibility_x0020_Criteria,
    Office365Outlook.SendEmail(
        "apurvanand.sahay@univaq.it",
        "Applied for the post " & Post_Applied.Selected.Name_x0020_of_x0020_the_x0020_Post,
        "My name is " & Name.Text & " And I have eligibility of " & EligibilityInfo.Text & ". Thanks."
    );
    SubmitForm(EditForm1),
    RemoveIf(
        [@Table1],
        EligibilityInfo.Text <> LookUp(
            Table2,
            Post_Applied.Selected.Name_x0020_of_x0020_the_x0020_Post = Name_x0020_of_x0020_the_x0020_Post
        ).Eligibility_x0020_Criteria
    )
)
```

FIGURE 4.6: Recruitment Drive in Microsoft PowerApps

In the example shown at the bottom of Fig. 4.6, the OCL-style language is used to specify the logical workflow of comparing the required eligibility with the application data, which are provided through a designed form. In case of the eligibility condition between the `Applicant` and the

`Recruitment Notification` table matches, the HR will be notified; otherwise, the current user input gets deleted. Then, the tables can be linked using the data connector used in the platform that connects different data stores such as Microsft SQL, Sharepoint, OneDrive, etc.

4.3.5 Salesforce Lightning

The Lightning platform[15] is the low-code answer to app development by Salesforce, a leader company in CRM technology. In Salesforce Lightning, workflows can be defined in four different ways, i.e., by using *Flows*, *Workflow field update*, *Process builder*, and with the help of *Apex code* [58]. In this paper, we focus on using *Flows*. *Workflow field update* and *Process builder* are limited in their ability to write workflow rules to create or delete records. In contrast, workflows through *Apex code* are supported only in the professional version of Salesforce Lightning, and it requires complex coding; thus, not favorable for inexpert users.

Core Modeling Elements: `Salesforce Lightning` platform has their BPMN `Event` element termed as *Start*, and *Stop*. The BPMN `Activity` element is instead supported by platform specific modeling elements such as *Add Object*, *Immediate Actions*, and *Scheduled Actions*. Finally, `Gateway` is referred as *Add Criteria* in the Lightning platform.

Workflow Modeling: New processes can be created when the user accesses the "Process Automation" function in the Salesforce Lightning platform. In addition, the business workflow template offers a way to link the action to an event. For example, an event can be "*a record changes*" for building a particular type of step when a record (row) changes in the Object (table). Or it can be processed by calling another process (also called the nested process). Also, the Flows mechanism is available in Salesforce Lightning to develop business logic through a drag-and-drop interface that provides the user with business modeling elements that include events, activities, and actions [100]. Finally, Apex coding can be employed to update fields during workflow executions. However, such languages are not used extensively by non-developers, and their adoption has pros and cons, as discussed in [58].

Data handling capabilities: In this platform, a table or an object can be created by using the `Object Manager`[16] that could be used to define the schema of various tables, which underpin the data of the application under development. Alternatively, a table can be imported from an external spreadsheet that allows manipulation only inside the Salesforce environment. Authentication is required to connect external spreadsheets/databases.

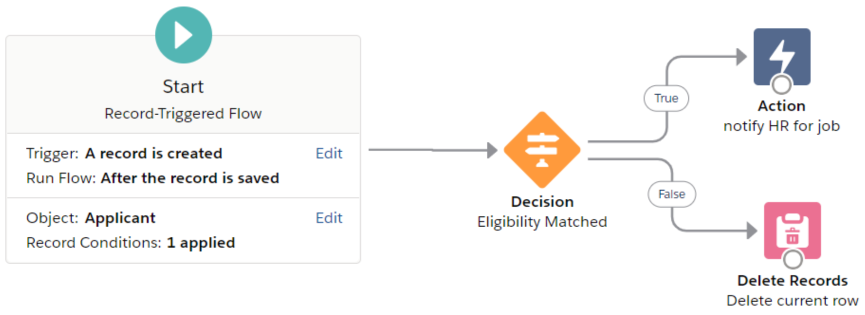


FIGURE 4.7: Recruitment Drive Workflow in Salesforce Lightning

The workflow shown in Fig. 4.7 depicts the *Flows* that get executed when a record is created in the `Applicant` table. The condition is set to equate the table `Applicant`'s eligibility information with the post applied to the row with the same post name in the table `Recruitment Notification`'s field of eligibility criteria. The eligibility condition is specified in the decision activity as follows.

[15] https://www.salesforce.com
[16] `https://help.salesforce.com/s/articleView?id=sf.extend_click_find_objectmgmt_lex.htm&type=5`

`!$Record.Eligibility_Information__c Equals !$Record.Post _Applied__r.Eligibility_Criteria __c`. If the condition is true, then the HR and Applicant will be notified about the applicant's success; otherwise, the current Applicant's row is deleted.

4.3.6 Thinkwise

Thinkwise[17] is a LCDP offering modeling tools available on desktop, Web, and mobile devices typically useful for large scale business software[18].

Core Modeling Elements: `Thinkwise` provides modelers with a BPMN-like notation to specify workflows as compared to the other LCDPs. The BPMN element `Lane` is defined by creating *user roles*, whereas `Events`, `Activities` and `Gateways` are defined by the *activity* elements that modelers can specify in a dedicated environment for specifying process flows. Also `Sequence Flow`, `Message Flow` and `Association` are supported to connect specified activities. All the above-mentioned modeling elements get executed by the Thinkwise execution environment. Several programming languages are supported to define business logic including SQL, R, Python, and Java.

Workflow Modeling: In the Thinkwise platform, process flows and business logic specifications are distinguished. A process action handles the process flow to create events, triggers, and decisions and their interactions via process steps [108]. In contrast, business logic is developed using the supported programming languages to handle data and execute controlled procedures. Furthermore, user roles can be defined to protect the application and enable its usage according to the user's role, such as developers, business analysts, users, etc. Along with intuitive graphical workflows, code-based business logic is also possible in Thinkwise, which supports different languages, including SQL, R, and Java.

Data handling capabilities: In Thinkwise, tables can be created by using the data model provided in the `Software Factory`[19] where the fields and data can be created or modified. Once the table of interest is created, data manipulations can be implemented through external databases such as DB2, SQL Server Management Studio, and Oracle. This means that the data handling occurs by interacting with the instance of the used external databases.

The Thinkwise data model related to the running example is shown in Fig. 4.8, whereas the related process flow is depicted in Fig. 4.9.

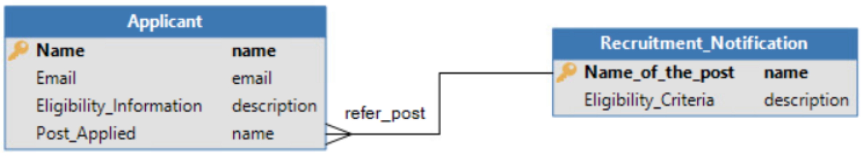


FIGURE 4.8: Recruitment Drive Data Model in Thinkwise

The SQL code implementing the business logic of the considered case study is shown in Listing 4.1.

LISTING 4.1: Logic for the case study Recruitment Drive

```sql
SELECT a.Name, a.Email, a.Eligibility_Information, a.Post_Applied
FROM Applicant a, Recruitment_Notification r
WHERE a.Post_Applied=r.Name_of_the_post AND
a.Eligibility_Information=r.Eligibility_Criteria
```

[17] https://www.thinkwisesoftware.com/
[18] https://www.thinkwisesoftware.com/en/overview/
[19] https://docs.thinkwisesoftware.com/docs/sf/sf_general.html

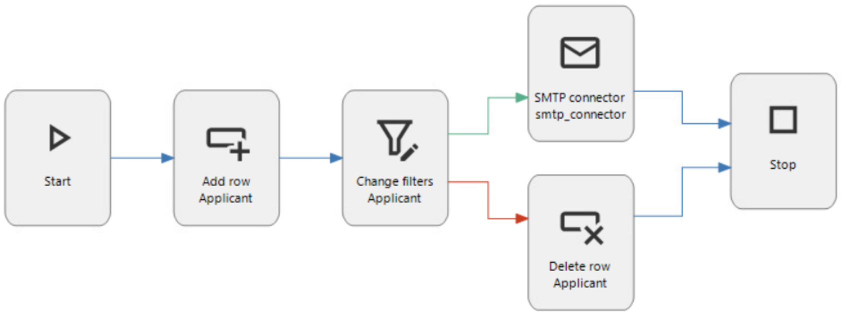


FIGURE 4.9: Recruitment Drive Process Flow in Thinkwise

Such a mixture of code-based and graphical-based business logic allows the business people and IT teams to collaborate and work in an agile environment to build an application. For example, the figure 4.9 depicts the business workflow that compares the eligibility information from the `Applicant` table to the eligibility criteria from the `Recruitment_Notification` table. If this condition is true, a success message is sent to the applicant/HR; otherwise, the applied person's record is deleted from the table `Applicant`.

4.3.7 Google Appsheet

Appsheet[20] is the LCDP from Google to build apps and support work automation.

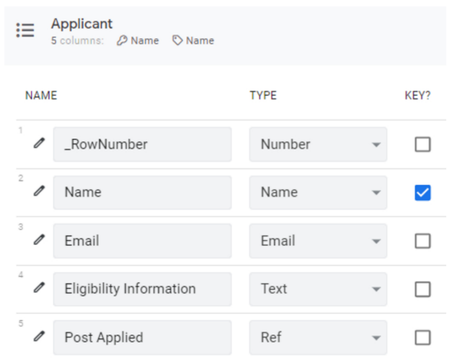
Core Modeling Elements: In Google Appsheet, the BPMN `Event` element is mapped to events that are categorized as "*New Workflow Rule*", "*When this happens*" and "*If this is true and Do this*". `Activity` is broadly termed as *Action Reaction* that acts on different types of objects, including Email, Notification, SMS, TakeAction, Webhook, and MakeDoc. Lastly, `Gateway` can be mapped to the events and conditions declared under the events "*When this happens*", and "*If this is true and Do this*".

Workflow Modeling: Workflow automation in Google Appsheet comprises three types of actions [53]. The first type is based on a data change that triggers an event when data is captured through the built application. The second type is scheduled on a specific date/time, and the third type is based on webhooks that can be scheduled or activated on data's additions/updates/deletions. Google Appsheet supports the connection to external services, including Zapier and IFTTT. Pre-built workflows are also available in Appsheet that can be reused to specify a newer workflow for a different app.

Data handling capabilities: Tables can be imported from external data sources from different vendors such as Google, Microsoft, Dropbox, Smartsheet, and Airtable. The data source is directly linked to the platform. This means that the data is defined (attributes, relationships, references, keys) and manipulated (search, filter, add, delete, etc.) by the spreadsheet formulae or by defined process workflows on Appsheet that sync with the used database.

In Appsheet, both tables `Applicant` and `Recruitment Notification` were defined and imported from Google Sheet, i.e., the spreadsheet tool provided by Google. The workflow is implemented by selecting the *Behavior* menu in the Appsheet Web development tool (see left of Fig. 4.10). Then, the current applicant's data in the `Applicant` table is chosen by selecting the "Target data" field in the form. On Such a table, the developer can add and update events such as mentioning the business logic `[Eligibility Information]=[Post Applied].[Eligibility criteria]` as specified in the field condition of Fig. 4.10. If this condition is true, then the necessary task(s) is performed. In our case, the task is to send the email to HR/applicant. Alternatively, if the condition

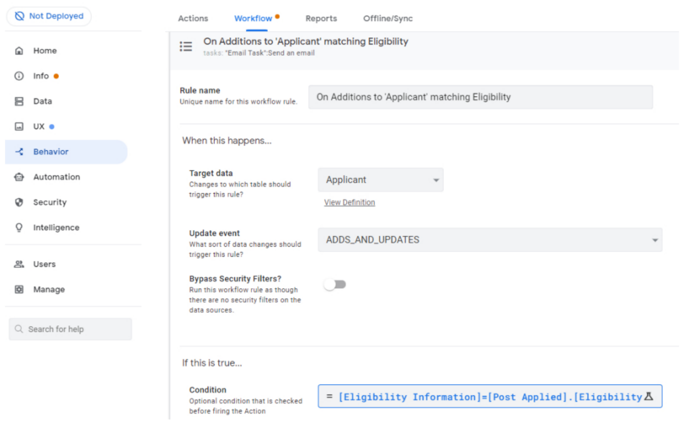
[20] `https://www.appsheet.com/`



Applicant
5 columns: Name Name

NAME	TYPE	KEY?
_RowNumber	Number	☐
Name	Name	☑
Email	Email	☐
Eligibility Information	Text	☐
Post Applied	Ref	☐

(A) Applicant Table



On Additions to 'Applicant' matching Eligibility
tasks: "Email Task":Send an email

Rule name
Unique name for this workflow rule. On Additions to 'Applicant' matching Eligibility

When this happens...

Target data
Changes to which table should trigger this rule? Applicant
View Definition

Update event
What sort of data changes should trigger this rule? ADDS_AND_UPDATES

Bypass Security Filters?
Run this workflow rule as though there are no security filters on the data sources.

If this is true...

Condition
Optional condition that is checked before firing the Action `= [Eligibility Information]=[Post Applied].[Eligibility`

(B) Business logic

FIGURE 4.10: Recruitment Drive Process Flow in Google Appsheet

`[Eligibility Information]<>[Post Applied].[Eligibility criteria]` is applied (Here, '<>' means *is not equal to*), then the delete action is performed on the whole row managed by AppSheet that leads to the deletion of the row back to the Google Sheet.

4.3.8 Amazon Honeycode

Honeycode[21] is the LCDP released by Amazon in June 2020 to support the development of Web and mobile applications.

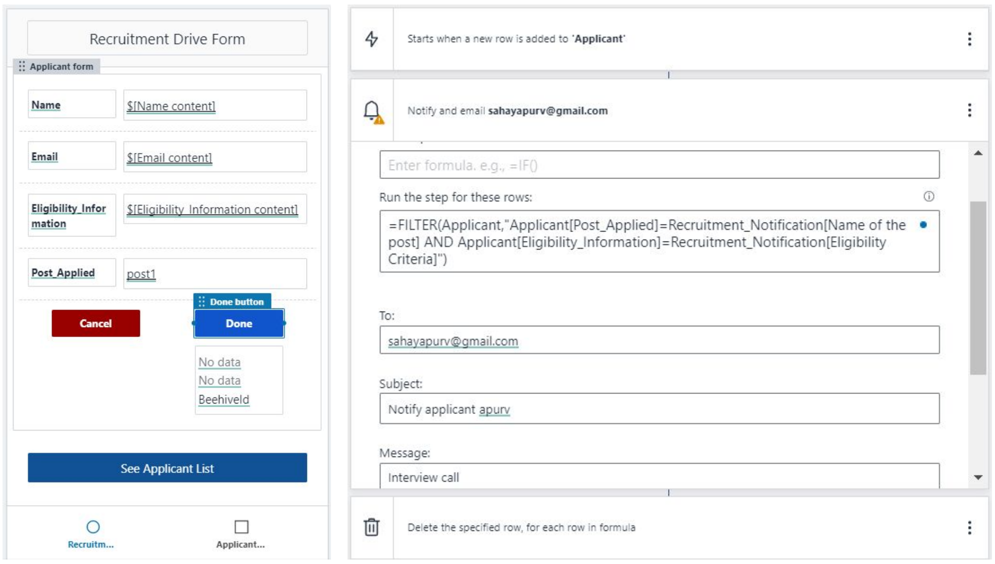
Core Modeling Elements: The Amazon Honeycode platform has their `Event` termed as *Start when clicked*. An `Activity` can be mapped to spreadsheet formulae to perform an action, e.g., *Add row to* and *Take data from and write to*. Finally, the `Gateway` can be implemented using platform specific SQL-like statements that include *filter* and *if* conditions.

Workflow Modeling: Amazon Honeycode uses spreadsheet data to create an application. An application is created by defining a spreadsheet which can then be transformed into list or report elements. Such elements show all the items in the built application along with a detailed view of an individual data element of the application and allow the user to use the form screen to give input to the application [8]. Further, the application's business logic is managed by spreadsheet formulae, which can add necessary actions such as notify, add a row, delete a row, overwrite an element in the screen, update or insert a row, and navigate another screen.

Data handling capabilities: Honeycode manages tables by importing CSV files or creating a blank table directly following the platform's wizard. Any data manipulation (add row, filter, search, etc.) can be done using Excel-like formulae within the platform.

In Amazon Honeycode, the tables `Applicant` and `Recruitment Notification` are created, followed by their respective screens (forms and list). At the submit button of the form, the condition specified (right-hand side of Fig. 4.11) is used to validate the applicant's job application by matching the input Eligibility_Information with the Eligibility_Criteria for that post mentioned in the `Recruitment Notification` table. If this matching is true, then the HR/applicant is notified. Alternatively, if we use inequality (<>) in Applicant's eligibility information with Recruitment Notification's eligibility criteria, we can delete the entire row. This will allow only those applicants to be considered for the chosen post with the required eligibility criteria as mentioned in the script at the bottom right side of Fig. 4.11.

[21] https://www.honeycode.aws


Recruitment Drive Form

Applicant form

Name: $[Name content]

Email: $[Email content]

Eligibility_Information: $[Eligibility_Information content]

Post_Applied: post1

Done button

Cancel | Done

No data
No data
Beehiveld

See Applicant List

Recruitm... | Applicant...

Starts when a new row is added to '**Applicant**'

Notify and email **sahayapurv@gmail.com**

Enter formula, e.g., =IF()

Run the step for these rows:

```
=FILTER(Applicant,"Applicant[Post_Applied]=Recruitment_Notification[Name of the post] AND Applicant[Eligibility_Information]=Recruitment_Notification[Eligibility Criteria]")
```

To: sahayapurv@gmail.com

Subject: Notify applicant apurv

Message: Interview call

Delete the specified row, for each row in formula

FIGURE 4.11: Recruitment Drive Process in Amazon Honeycode

4.4 Discussing LCDPs with respect to BPMN modeling constructs

Table 4.3 shows the coverage of the BPMN constructs overviewed in Table 4.1 from the previously analyzed LCDPs.[22] According to the performed analysis, depending on the adopted LCDP, the specification of workflow processes can or cannot follow a BPMN-style. For example, low-code platforms such as OutSystems, and Mendix provide users with a platform-specific BPMN-style modeling framework. In contrast, Microsoft PowerApps, Zoho Creator, Google Appsheet, and Amazon Honeycode support a non-BPMN-style modeling framework that uses proprietary coding or a hierarchical workflow mechanism to define business flows. Other platforms such as Salesforce Lightning and Thinkwise support both BPMN and non-BPMN styles.

TABLE 4.3: BPMN workflow constructs covered by the analysed Low-Code Platforms

BPMN Elements	OutSystems	Mendix	Zoho Creator	PowerApps	Lightning	Thinkwise	Appsheet	Honeycode
Swimlanes								
Pool								
Lane						✓		
Flow Object								
Event	✓	✓	✓	✓	✓	✓	✓	✓
Activity	✓	✓	✓	✓	✓	✓	✓	✓
Gateway	✓	✓	✓	✓	✓	✓	✓	✓
Data								
Data Object		✓	✓	✓			✓	
Data Store			✓	✓	✓	✓	✓	✓
Message	✓	✓						
Artifacts								
Group								
Annotation	✓	✓						
Connecting Objects								
Sequence Flow		✓			✓	✓		
Message Flow					✓	✓		
Association		✓			✓	✓		

[22]The complete analysis is available at `https://docs.google.com/spreadsheets/d/1lllnNTwXNgQCXPVbaRBAk_dIs6gPVdOCNxub_MBoq-w/edit?usp=sharing`

TABLE 4.4: Main mechanisms to specify workflows in low-code development platforms

	Code	Hierarchical Structure	Automated BPMN	User-Defined BPMN	Hybrid
OutSystems				✓	
Mendix				✓	
Zoho Creator	✓				
Microsoft PowerApp	✓	✓			✓
Salesforce Lightning	✓		✓	✓	✓
Thinkwise	✓			✓	✓
Google Appsheet		✓			
Amazon Honeycode	✓				

The combination of process and data defined and managed with an intuitive visual interface with minimal coding describes the peculiar characteristics of any LCDP, which is supposed to support the fast development and management of business applications. In Section 4.3, the considered LCDPs are further discussed by considering for each of them the BPMN core elements covered and the mechanisms provided to model workflows and handle corresponding data. Based on the discussed core modeling elements across several LCDPs and the content of Table 4.3, it is noticeable that although there may not be one-to-one similarities between the BPMN and the LCDPs modeling elements, an LCDP is capable of expressing workflows (similar to BPMN) and the associated business logics in different ways while developing business applications.

TABLE 4.5: Supported data management

	Data Import	Data Sync
OutSystems	✓	
Mendix	✓	
Zoho Creator	✓	✓
Microsoft PowerApp		✓
Salesforce Lightning	✓	
Thinkwise		✓
Google Appsheet		✓
Amazon Honeycode		✓

To make an overview of the different types of business workflows that might be specified with the analyzed platforms, we can distinguish three ways of defining business processes: *i)* use of drag-and-drop BPMN-like tools to create business flows and logic, *ii)* non-BPMN-like tools that use either some programming language or some hierarchical decision-making structure, *iii)* combination of the first two means known as a hybrid category. Thus, as shown in Table 4.4, we can distinguish five main mechanisms to specify workflows in LCDPs as described in the following:

1. *Code-defined modeling:* A business process is created using a proprietary or a known programming language such as Java, Javascript, SQL, etc. LCDPs such as Zoho Creator and Salesforce Lightning Platform support proprietary Deluge code and Apex, respectively. Also, Thinkwise supports multiple languages such as SQL, R, Java, etc., to define business logic.

2. *Hierarchical structure of business process:* A business process is created with a tree-shaped business flow, causing one event to follow another based on some conditions or trigger points. LCDPs such as Google Appsheet or Microsoft Power Automate in PowerApps support a hierarchical-based structure for business flows.

3. *Automated BPMN-like structure:* A business process is created by an automatically defined BPMN structure where the user can fill the events, objects, or conditions based on the predefined BPMN flow. Salesforce Lightning Platform supports automated and user-friendly specifications of complex business modeling.

4. *User-defined BPMN modeling:* A business flow is created by a BPMN-like experience defined by the user based on her requirements. LCDPs such as OutSystems, Mendix, Thinkwise, etc. support BPMN-like modeling that expresses an extensive range of business flows.

5. *Hybrid modeling:* Business flows are partially implemented using two or more listed business process types. LCDPs such as Salesforce Lightning platform, Microsoft PowerApps, and Thinkwise support multiple workflow mechanisms for the same working business flows. Salesforce Lightning platform uses automated BPMN and code-based modeling in Apex language. Microsoft PowerApps uses a hierarchical structure in Microsoft Automate and OCL-based coding. Thinkwise uses a user-defined BPMN for the business flow and SQL coding for the business logic.

Concerning the way data are managed by LCDPs, it is possible to categorize them in two ways as shown in Table 4.5 and explained below.

1. *Data Import*: LCDPs can import data structures from external sources such as Microsoft Excel, Azure, SQL, etc. In this category, all the structural definitions of the data table, such as attributes and relationships, are imported into the considered LCDP, and they are transformed into the host LCDPs structures to create pages such as forms, lists, etc. Thus only the table's design is imported, not the corresponding instances. No direct manipulation can be done in the initial data source without using the APIs to integrate them. LCDPs such as Zoho Creator, Salesforce Lightning, Outsystems, etc., are in this category.

2. *Data Sync*: LCDPs can retrieve and store data from and to the considered data sources like Microsoft Excel, Azure, SQL, etc. LCDPs such as Microsoft PowerApps, Google Appsheet, Amazon Honeycode, etc., import the whole instance of the considered database, and data manipulations are synced back with the initial data source.

4.5 Discussing LCDPs with respect to BPMN quality criteria

This section discusses the quality assessment of LCDPs by relying on existing approaches for BPMN. In particular, nine quality criteria characterize, according to [49], the quality of a modeling language: expressiveness, understandability, conformity to standards, granularity, executability and orchestability, measurability, business rules, supporting tools, and validation in real environments. In the following, we discuss such criteria to analyze the quality of the eight considered LCDPs:

Semantic richness and expressiveness: BPMN2.0 defines process models based on core elements such as flow objects, data objects, artifacts, and connecting objects. It is noticeable from Table 4.3 that all the core modeling elements of BPMN can be interpreted in the LCDP-specific modeling constructs. Such modeling constructs allow a developer to build a workflow for an application that could be executed by applying the business logic of that application. For example, the case study `Recruitment Drive` is modeled using BPMN in section 4.1 and its automatized part built by various LCDPs were described is section 4.3.

Understandability: The understandability aspect of BPMN is described in [32] in which many modeling guidelines are considered, such as validating models, minimizing model size, applying hierarchical structure with sub-processes, and other optimum use of different modeling elements. These guidelines suggest design decisions that allow the modelers to choose the most relevant modeling tool according to their modeling purpose, such as process learning, information system development, etc. Compared to the understanding of BPMN, all the eight considered LCDPs' workflow mechanisms are categorized and highlighted in Table 4.4. Low-code platforms such as Zoho Creator, OutSystems, and Salesforce Lightning support a proprietary code-based and user-defined BPMN-like modeling workflow mechanism. The workflow is designed in an easy-to-use built-in modeling mechanism that could either use BPMN-like modeling constructs such as in Salesforce Lightning or use a combination of different hierarchical or BPMN-like or code-based *hybrid* modeling constructs. The combination of varying modeling constructs in LCDPs helps the developer build an extensive enterprise-grade application. It can use hierarchical-based modeling constructs and encode the modeling workflow using some traditional programming languages like SQL, Java, R, etc. This combination of workflow constructs would enable the developer to use hierarchical-based workflow for more straightforward tasks and fill up the remaining modeling workflow requirements using code technologies.

Conformity to standard: BPMN2.0 is standardized by ISO/IEC 19510[23]. The International Standard defines four conformance types: Process Modeling Conformance, Process Execution Conformance, BPEL Process Execution Conformance, and Choreography Modeling Conformance. In contrast to BPMN, all the considered LCDPs are vendor-locked development platforms. Therefore, these LCDPs have their proprietary workflow mechanisms, although some of the LCDPs, such as OutSystems, and Mendix, have motivated their workflow mechanisms from the BPMN modeling construct standard.

Granularity: BPMN models can be modularized by using swimlanes that are divided into pools and lanes. Moreover, a BPMN model can be granularized according to three different modeling details that are useful for various stakeholders: *i)* defining a process flowchart, including the overall activities and high-level decision conditions, is helpful for the end-user to understand; *ii)* describing the whole process extensively on different roles of process, data, information, etc., may be beneficial for developers.; *iii)* explaining the modeling flowchart that comprises sufficient information such that the process may be defined for analysis and simulation. This detailed flowchart can then be executed to develop some code. The analyst uses this level of detail and is further handed over to the developer for proper execution in compliance with the detailed process model.

Compared to BPMN, LCDPs have separated workspace to specify domain (data) models and for modeling user interfaces and processes. Also, some LCDPs such as Microsoft Powerapps, Amazon Honeycode, and Google Appsheet may support an integrated tool to process models within the user interface that can manipulate the data model designed for an application. Microsoft PowerApps does support OCL-based scripting inside the pages of the application along with the external usage of the Power Automate platform to granularize the process model from different pages of the application. The granularization in an LCDP can be divided into three parts:

– *Separation of process participants:* The entire process is built across different organizations; then, it is difficult to maintain consistency as there will be interoperability and

[23]`https://www.omg.org/spec/BPMN/ISO/19510/PDF`

authentication issues. For example, a company's recruitment process may require payment from an external payment gateway service done by the applicant. This means that the modeling construct built in a particular LCDP has to handle both of these processes within their platforms. For instance, the import module or the API-enabled feature in Mendix is used to interoperate with external services, which expert developers use to handle the processes of two distinct participants.

- *Separation of the roles of process participants:* The process built in an organization to support multiple stakeholders needs to be authenticated. Different user roles such as analysts, developers, users, etc., need to be filtered according to their roles and hierarchy within that organization. Such roles must be defined within the modeling construct of an LCDP. For example, the user roles in Thinkwise allow different organizational stakeholders to access a specific part of the application according to their defined roles.
- *Separation of processes, data, and user-interfaces:* Although the process modeling, data handling, and user-interface building can be overlapped to develop an application, it gives a more precise overview of the application if these three parts are separated. This allows the application to be agile with minimum coupling across these three parts allowing the developer to test and deploy the application. For example, OutSystems supports the separation of process, data, and user interface, which increases the application's maintainability as it could easily manage the consistency available in these three parts of the application development.

Executability and orchestability: BPMN Process Execution Conformance type is mandated in the BPMN execution tool that interprets models to the operational semantics. This tool supports the BPMN mapping to WS-BPEL (Web Service-Business Process Execution Language) [81]. BPMN Choreography Conformance is also implemented in BPMN packages. The package includes BPMN core elements, choreography diagrams, and collaboration diagrams that include pools and message flow. Since LCDPs are cloud-based application development platforms that mainly use models to develop an application, the process modeling constructs within an LCDP are executed either by code generation or by directly compiling models at run time [105]. Therefore, all the process modeling workflows within an LCDP are executed according to their execution engine. For example, some code-generated LCDPs are OutSystems, Thinkwise, Nintex Workflow, etc., while some LCDPs having compiled models to be executed at run-time are Mendix, Zoho Creator, Microsoft PowerApps, Salesforce Lightning, Amazon Honeycode, and Google Appsheet.

Measurability: Some aspects of BPMN2.0, such as graphical notations or metamodel quality, cannot be compared thoroughly with measurable quantities. However, automation and execution of BPMN models can be measured with a mostly one-to-one mapping of elements to their corresponding code. Therefore, BPMN supports partial measurability. For LCDPs, as all the process models must be either code generated or directly executed at run-time, each modeling element can be measured in terms of memory space and execution time. Therefore, process modeling can be measurable regarding execution time and the amount of required memory.

Business rules: BPMN supports a business rules engine that takes input from a process and produces the intermediate and final output. For example, a specified task gets executed by a business process engine. In LCDPs, inputs from multiple pages (such as forms or reports) are given to the process modeling used in developing an application. Also, multiple outcomes or tasks (such as scheduling tasks, approval tasks, etc.) can be executed using the designed process model within an LCDP. Therefore, all LCDPs support business rules.

Supporting tools: Some of the most used tools that support BPMN are Drawio, Microsoft Visio, Lucidchart, Visual Paradigm, SmartDraw, etc. All the vendor-locked LCDPs support their built-in modeling tool.

Validation in real environments: Many vendors support the validation tools for the BPMN models, e.g., viadee Process Application Validator in Camunda[24], BPMN validation tool in WebRatio[25], etc. Also, another application named bpmnlink[26] validates the BPMN diagrams based on configurable rules. These tools validate the syntactically BPMN modeling elements using configurable rules developed by programming languages such as Java. LCDPs support complete testing environments where developers can test applications and validate them. As the LCDP is either code generated or directly compiled on models, the validator is built using the concepts of model-driven engineering [97].

With the listed quality standard/criteria of modeling languages mentioned above, different supported modeling constructs within a LCDP are considered. These considered modeling workflow guides the low-code application developer to choose a particular process modeling mechanism that is best suited for the application to be built.

Another quality criterion to consider when selecting LCDPs is the availability of automation of process modeling through Artificial Intelligence (by using Robotic Process Automation [2]) and machine learning techniques that can predict and suggest the following probable best action(s) based on the previous workflow used in a similar application. If a process construct in an LCDP builds a process workflow, the construct may suggest the next probable step(s) based on the learning from similar workflows. For instance, the OutSystems AI assistant works through the business workflow, predicts the next step, and recommends the best option available to configure and adapt a business logic according to the appropriate context[27]. Similarly, in Microsoft PowerApps and Power Automate, AI Builder is used to help businesses to use AI through a point-and-click experience by building a custom model or choosing a pre-built model to train it and then using insights from that AI model[28]. These features go in the direction of Intelligent Modeling Assistants [77].

Threats to validity

In this section we discuss the threats to validity by dividing them in intrinsic and extrinsic.

Intrinsic threats Some aspects might represent threats while evaluating data and business flows of different LCDPs. The free version of the considered LCDPs is used throughout the study. This means that some features, such as using APIs, collaboration features, etc., are not experimented with in handling external data sources. The study elaborates on the accessible version of the LCDPs, which focuses primarily on the direct use of external data without much coding or using different APIs. Also, there is no direct correspondence between the functionality of BPMN core elements and the LCDPs elements. To tackle this threat, BPMN elements' functionality is partially matched with the LCDPs elements. For example, the data model in Mendix handles the creation of data and their associated connecting objects with one another in a graphical manner. This data handling can only be represented as a separate entity apart from their connecting objects within the same or different swimlanes used in the BPMN2.0 specification.

[24]`https://camunda.com/blog/2017/10/viadeeprocessapplicationvalidator/`
[25]`https://methodandstyle.com/bpmn-validation-tool-improved-by-webratio/`
[26]`https://github.com/bpmn-io/bpmnlint`
[27]`https://www.outsystems.com/ai/`
[28]`https://docs.microsoft.com/en-us/ai-builder/overview`

Extrinsic threats Some extrinsic factors that can threaten understanding the different kinds of data and business flow apart from the discussed LCDPs are as follows. The study focuses on 8 LCDPs only, and many other LCDPs, such as Kissflow, ZappDev, Nintex, etc., are not considered. However, we decided to focus the analysis by considering leading platforms that are niche players according to the Gartner report [117].

4.6 Summary

Business process management in application development plays a crucial role in improving business performance. The developer must create workflows involving several services to make the task agile and lean. In this chapter, we described the usage of modelling frameworks that includes various process modeling along with the data-handling capabilities of LCDPs. The modelling constructs of various LCDPs are discussed with respect to those of BPMN. These modelling constructs are categorized, and the execution of these constructs of different LCDPs is explained with the help of an illustrative case study.

Therefore, an LCDP is considered a visual platform that allows developers, including those without programming background and knowledge, to create various software systems. In a low-code platform, developers can leverage pre-existing models and components to combine and reuse them, enabling the creation of new applications. This approach offers flexibility and ease of use to citizen developers, empowering them to participate in software development processes.

One way to compose models in an LCDP is through chains of model transformations. However, with multiple available chains, it becomes crucial to select the most suitable ones based on different quality criteria. In the next chapter 5, we delve into the identification of various transformation chains and discuss the process of selecting the optimal chain. This enables developers to efficiently find the best possible ways to achieve the desired output model, enhancing the overall effectiveness and productivity of the low-code development process.

Chapter 5

Identifying optimal model transformation chains

The process of chaining model transformations can involve multiple steps, each of which transforms the model in a specific way. Identifying the correct model transformation chain is crucial for ensuring that the final model is accurate and meets the desired specifications. In order to chain the transformations between the source and the target model, we need to identify all possible transformations in between that takes the source metamodel of the transformation and then chain them with different metamodels which is further continued till the target metamodel is achieved at the end of the transformation chain.

A motivating example of the need for identification of model transformation chain is in the development of a complex software system. The system may be designed using multiple modelling languages and tools, each with their own strengths and weaknesses. For example, an initial design may be created using UML (Unified Modeling Language), but later stages of development may require the use of a different language, such as SysML (Systems Modeling Language), to better model the system's behavior. In order to successfully transition from one language to another, a series of model transformations must be applied.

The identification of the correct model transformation chain is not always straightforward, as there may be multiple options available, each with their own trade-offs. For example, a transformation that preserves more of the original design may be more complex and time-consuming to implement than one that simplifies the design but loses some information. Identifying the correct model transformation chain requires a thorough understanding of the requirements and constraints of the system, as well as an understanding of the capabilities and limitations of the available modeling languages and tools. Overall, identifying candidate model transformation chains is an important step in ensuring that the final model is accurate and meets the desired specifications. It requires a thorough understanding of the system, the modeling languages and tools, and the trade-offs involved in each possible transformation.

This chapter dedicates the approach towards identifying possible model transformation chains from a model repository/folder. After the identification process, the chapter also focuses on the next logical step, selecting the optimum transformation chain(s) using search-based optimization techniques.

5.1 Automated identification of model transformation chains

Identification of possible model transformation chains can be made using the following steps. The first step is to check if the direct model transformation exists that transforms the source model to the target one. This is the base condition. Then we identify all the available model transformations that transform the source model into any of the available intermediate models. Further, we reuse the intermediate model as the source model with the help of base conditions (using depth-first

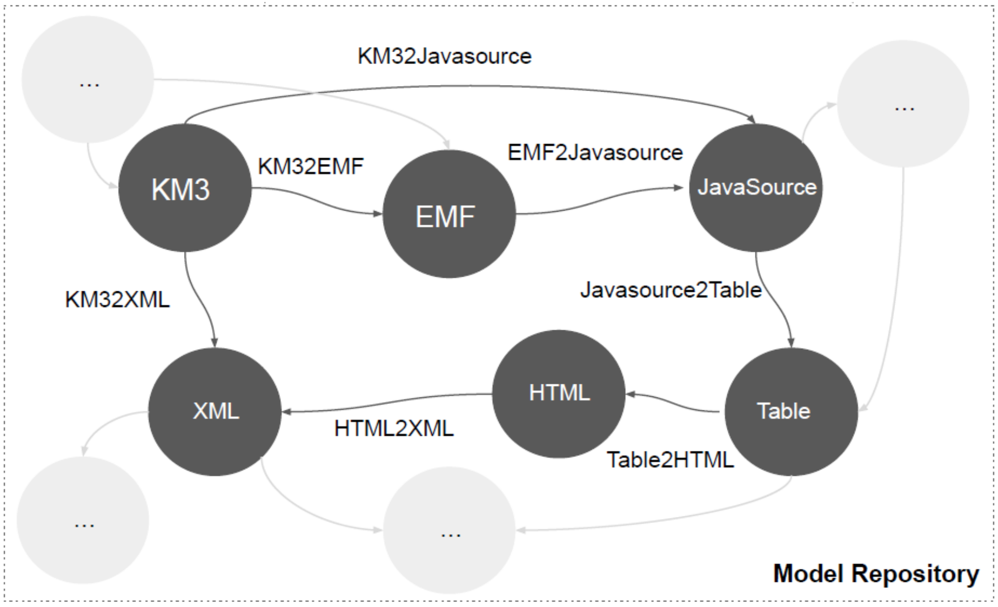


FIGURE 5.1: Model transformation chaining scenario 1

search recursively), and we repeat this step till we reach the target model. Then, we check all the available model transformations that transform the identified intermediate model into the target model. Lastly, the source, intermediate and target models are composed and returned as a model transformation chain. The code for identifying transformation chains and executing them is given in the GitHub repository available at `https://github.com/lowcomote/chain-optimisation.git`. The pseudocode for identifying the possible model transformation chains is given in the Algorithm 1.

The input for identifying possible transformation chains is the source and target models along with the source and target metamodels. Figure 5.1 shows an explanatory scenario consisting of different metamodels and model transformations. Each node in the figure represent an ecore metamodel while the directed edge between them repesent a model transformation. The metamodel nodes are KM3.ecore, EMF.ecore, JavaSource.ecore, Table.ecore, HTML,ecore and XML.ecore. If KM3.ecore, and XML.ecore are given as input to the previously mentioned procedure, which is encoded in Algorithm 1, the output would consist of lists of chained ecore models (metamodels), i.e., [[KM3.ecore, Ecore.ecore, JavaSource.ecore, Table.ecore, HTML.ecore, XML.ecore], [KM3.ecore, JavaSource.ecore, Table.ecore, HTML.ecore, XML.ecore], and [KM3.ecore, XML.ecore]. We can represent the following model transformation chains as follows.

CH_1: $KM3 \rightarrow XML$

CH_1: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

CH_3: $KM3 \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

Another example of available metamodels and transformations in a folder/repository is shown in Fig 5.2. All possible paths between a source metamodel and a target metamodel by applying Algorithm 1 are as follows:

CH_1: $KM3 \rightarrow XML$

CH_2: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

CH_3: $KM3 \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

Algorithm 1 Pseudocode for identifying model transformation chains

```
1: procedure IDENTIFYCHAIN(sourceMM, targetMM)
Require: $ETL$ = etl transformations
                         ▷ returns true if a tranformation transforms sourceMM to targetMM
2:     findEtl(sourceMM, targetMM)
3:     if findEtl(sourceMM, targetMM) is true then
4:         Store sourceMM, targetMM in an ArrayList A1
5:     end if
6:     if findEtl(sourceMM, targetMM) is false then
7:         Traverse the folder/repository that contains metamodel files
8:         Store each metamodel as an intermediate metamodel MM_inter and give a name to its
           model M_inter same as the name of the metamodel
9:         Initialize visited[] array
10:        if findEtl(sourceMM, MM_inter) is true then
11:            IdentifyChain(sourceMM, MM_inter)
12:            Store MM_inter in the ArrayList A2
13:            visited[MM_inter, index] = true
14:            sourceModel = M_inter
15:            sourceMM = MM_inter
16:        end if
17:        visited[MM_inter, index] = false
18:        if findEtl(MM_inter, targetMM) is true and visited[MM_inter, index] = true then
19:            Store targetMM in ArrayList A2
20:        end if
21:        Add A1 and A2 in the list of ArrayList A3 and remove duplicates, if any.
22:    end if
23:    Return list of Arraylists of chained transformations that can transform from sourceMM to
       targetMM
24: end procedure
```

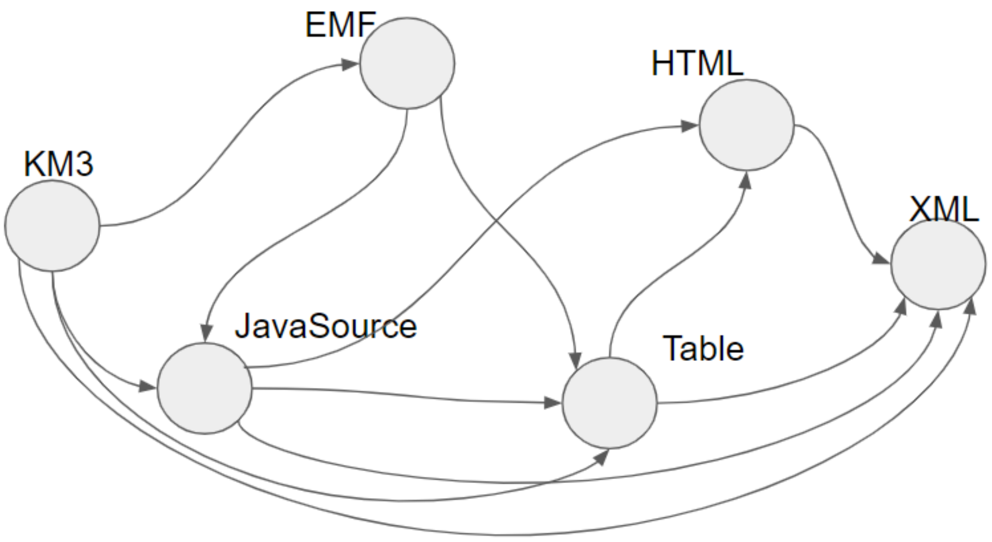


FIGURE 5.2: Model transformation chaining scenario 2

CH_4: $KM3 \rightarrow JavaSource \rightarrow HTML \rightarrow XML$

CH_5: $KM3 \rightarrow EMF \rightarrow Table \rightarrow HTML \rightarrow XML$

CH_6: $KM3 \rightarrow Table \rightarrow HTML \rightarrow XML$

CH_7: $KM3 \rightarrow Table \rightarrow XML$

CH_8: $KM3 \rightarrow JavaSource \rightarrow Table \rightarrow XML$

CH_9: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow HTML \rightarrow XML$

CH_{10}: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow XML$

CH_{11}: $KM3 \rightarrow JavaSource \rightarrow XML$

CH_{12}: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow Table \rightarrow XML$

CH_{13}: $KM3 \rightarrow EMF \rightarrow Table \rightarrow HTML \rightarrow XML$

The goal of identifying model transformation chains is to understand the sequence of model transformations that can be applied to the model of interest to potentially generate artifacts conforming to the desired metamodel. The selection of the optimal transformation chain(s) among those identified chains is detailed in section 5.2, whereas the optimization of their execution is elaborated in Chapter 6.

5.2 Selecting optimal model transformation chains

In model repositories [17, 48], modelers can contribute to improving the reuse of modeling artifacts by sharing models and modeling artifacts with the community or with a company owning a repository infrastructure. This gives the opportunity to modelers to use modeling repositories and adopt model management tools as software-as-a-service [22]. Indeed, model transformations can be invoked as a solution to a given problem: given an input model and a required output metamodel, produce the output model transparently. This application is possible thanks to transformation composition. In large model repositories, multiple transformations can be composed in order to satisfy the user's request. This opens a possible problem, that can be summarized as selecting the best transformation chains, when multiple chains are available, according to the user-defined preferences or criteria. Transformation coverage [71] can be one of the possible quality criteria to consider in selecting transformation chains, as well as aspects affecting performance, e.g., the number of transformations involved in the chain. Existing works [19, 18] propose automated selection mechanisms based on pre-defined quality criteria, i.e. coverage or information loss. These criteria are hardcoded in the selection algorithm, although, to the best of our knowledge, there is no work with a flexible and customizable mechanism for defining selection criteria.

In this section, we propose to use MOMoT [44], a search-based model-driven framework, to obtain Pareto-optimal solutions in the chain selection problem, based on user-defined criteria. We evaluate the proposed approach by comparing the obtained results with the ones obtained in [19], and we show how the approach can scale in terms of execution time and other parameter configurations.

5.2.1 Background and Motivating Example of chain selection

When multiple transformation compositions, i.e., chains, are available in our setting computed in section 5.1, different criteria can be defined to characterize what transformation chains are evaluated as the optimal chain(s). Potentially relevant criteria are denoted as transformation coverage and information loss in [19], but others may be of interest as well, or trade-off solutions when criteria are considered simultaneously. An example of the transformation chaining problem is adopted from [19] and depicted in Fig. 5.3. Given a user request composed of the input model, i.e. Sample-km3, which conforms to the source metamodel KM3, and the required metamodel for the output, i.e. XML, the process of finding optimized chains can be summarized as follows.

First, a discovery phase starts, by inspecting the available transformations in the repository, which can also be a local folder, selecting the needed transformations and deriving the needed transformation chains to get the final result. In Fig. 5.3, steps involving identifying possible transformation chains are represented as filtering from the repository in which the transformations and the associated metamodels can be chained. The algorithm for identifying possible transformation chains is given in the Algorithm 1 and explained in section 5.1. In our scenario, the selected metamodel nodes and transformations are reported in grey, excluding the remaining repository in light grey.

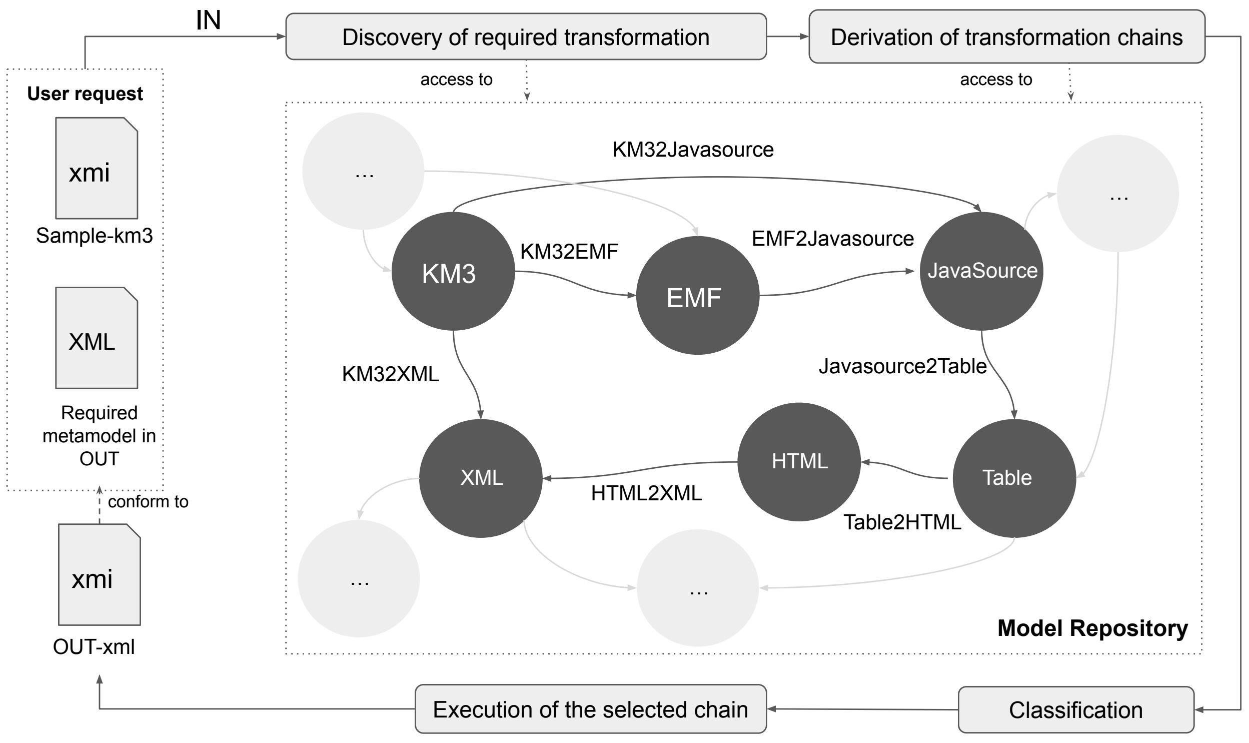


FIGURE 5.3: Model transformation chaining scenario

In this scenario, the list of available chains is composed of three available chains:

$Ch_1 : KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

$Ch_2 : KM3 \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

$Ch_3 : KM3 \rightarrow XML$

This list of available chains is given to the classification process that ranks the list according to the selected quality criteria or parameters and the selected chain is then executed. The execution of the chain returns the required model, i.e. the model conforming to the XML metamodel.

As said, the quality criteria that can be considered in these scenarios may be various. For instance, if we consider *i)* transformation coverage, *ii)* transformation complexity or *iii)* the number of transformation hops, we may have different results in the selection process. Transformation coverage is defined as the degree of completeness of a transformation which means how many metamodel elements (i.e., metaclass and attributes) of the source model to be transformed are consumed by the transformation [9]. This may affect the transformation process since, the more transformation covers constructs from the metamodel, the less result should lose output elements. The transformation complexity can be estimated by how much the rules, operators and expressions are used in the transformation chain. This is a new criterion to define a chain that is the best one by using a heuristic which describes the complexity of the transformation by counting the static elements of the transformation. This may affect the result in terms of performance, for example, since the more transformation engine has to interpret complex operations, the more slower the execution will be. When the repository contains large graphs of available transformations, the user may be interested in getting the result faster. The third criterion considered is the number of transformation hops that are used to achieve the target model. This is calculated as the width of the graph visits for each transformation chain. Again, the performance may be affected, since reducing the number of hops may reduce the chance of encountering a bottleneck transformation. In this case it would make sense to combine the coverage and complexity with this criteria, so that the selection becomes a multi-criteria optimization problem.

The problem of selecting the optimal chains according to different criteria is an open issue that has been partially covered in [19, 18], where these criteria are hard-coded in algorithms implementing the classification mechanism. This results to be limiting, first for the effort required to add new criteria in the evaluation, second for the lack of support in defining an optimization strategy considering multiple criteria.

To overcome these limitations, in the next subsection, we present how we have used MOMoT (Marrying Search-Based Optimization and Model Transformations), a model-driven optimization framework [27] using search-based techniques to optimize different criteria and, therefore supports selecting optimal transformation chains.

MOMoT[1] is a framework that uses MDE principles to solve complex multi-objective problems by using search-based optimizations. The problems are represented as Ecore metamodels, and in particular, instances of the metamodel are used to solve specific problems. Such instances are manipulated by an in-place graph model transformation expressed in Henshin [15]. To that end, the framework targets at optimal transformation sequences leading to optimal models rather than direct model manipulations. The output model is characterized by different constraints and objectives which are written in OCL or a Java-like expression language (Xbase) [27]. Finally, a sequence of transformations and parameters are executed, and the pareto-optimal solution is found by using search-based optimization [45] that includes different algorithms such as Random Search, NSGA-II, NSGA-III, etc., which are defined in the MOEA[2] framework.

5.2.2 Proposed approach of selecting transformation chain: MOMoT

In Fig. 5.4 we outline how we use MOMoT to support the selection of optimal chains. It shows the workflow used to run the selection process, starting from the definition of the needed artifacts, the configuration of the existing MOMoT modules, and the obtained results. In the following, we walk through the process and describe the artifacts used following the labels in Fig. 5.4. As elaborated in section 5.2.1, we anticipate the user input to include an input model and the required output metamodel. In addition, the repository that will be analyzed by looking for chains is also given as a parameter. In our case, we use a local folder containing Ecore models (metamodels) and ETL transformations. ETL is the transformation language part of the Epsilon [63] framework that we have used to test the implementation, but the approach can be easily re-applied for other transformation languages, e.g., ATL [57].

[1]http://martin-fleck.github.io/momot/
[2]http://moeaframework.org/

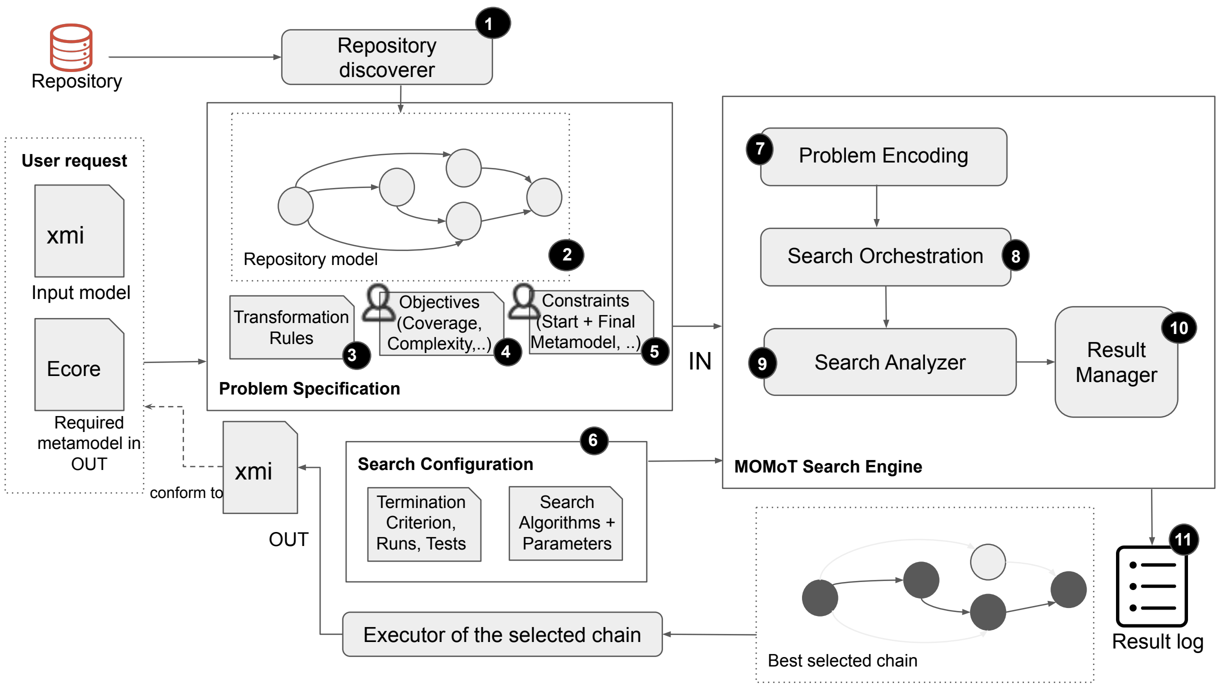


FIGURE 5.4: The proposed MOMoT extension for optimal chain selection

In the following paragraphs, we address the different parts of the problem specification shown in Fig. 5.4 and describe how optimal chain selection is supported with the individual components of MOMoT. First, the process for retrieving available transformations from the repository is described. We then present the metamodel, which describes the problem domain, hence carries elementary information of the chain selection task and maintains the current solution state, i.e., the transformation path with respect to the input model to be transformed. Afterwards, the transformation model used for assembling feasible chains through evolving the model-based problem representation is described. After that, the quality criteria we consider and evaluate in the chains are presented along with the validity conditions. Then, the search-based evaluation process configuration is given, including the algorithms used. Finally, the MOMoTs arrangements are discussed, including 1) the approaches to rewriting the problem model by rule applications to d

Repository Discoverer A repository/folder is first analyzed by the Repository discoverer ① which may be implemented based on the tools developed in [37] or programmatically finding out the possible chains as shown in the Algorithm 1 in section 5.1. This module analyzes the repository given as input in order to create the Repository model in ②, which is needed by MOMoT as part of the problem specification. We summarize the discovery strategy as follows: for each Ecore model, we programmatically create a node of the graph and for each transformation we add an edge. We highlight that ETL transformation modules do not include links to the source and target metamodels which are bonded at the configuration phase. For this reason, in order to run the discovery, we consider as pre-requisite that each ETL module must include in the header section the declaration of the source and target metamodels (this is optional in ETL but it simplifies the discovery phase). In this way the discoverer is able to retrieve the metamodels and set the edges of the graph. This is a requirement needed to use the static analysis [106] of the ETL transformation files which helps to find out the source and target metamodel of the transformation.

Modeling the problem in MOMoT MOMoT leverages a graph-based representation of the problem which it mutates by applying search strategies as a matter of exploring possible model variations that constitute the search space. Such a model for our purpose encompasses vital task details like discovered transformations in our repository model, the currently selected path to transform the input model, or transformation-related metrics. Therefore, we propose the metamodel reported in Fig. 5.5.

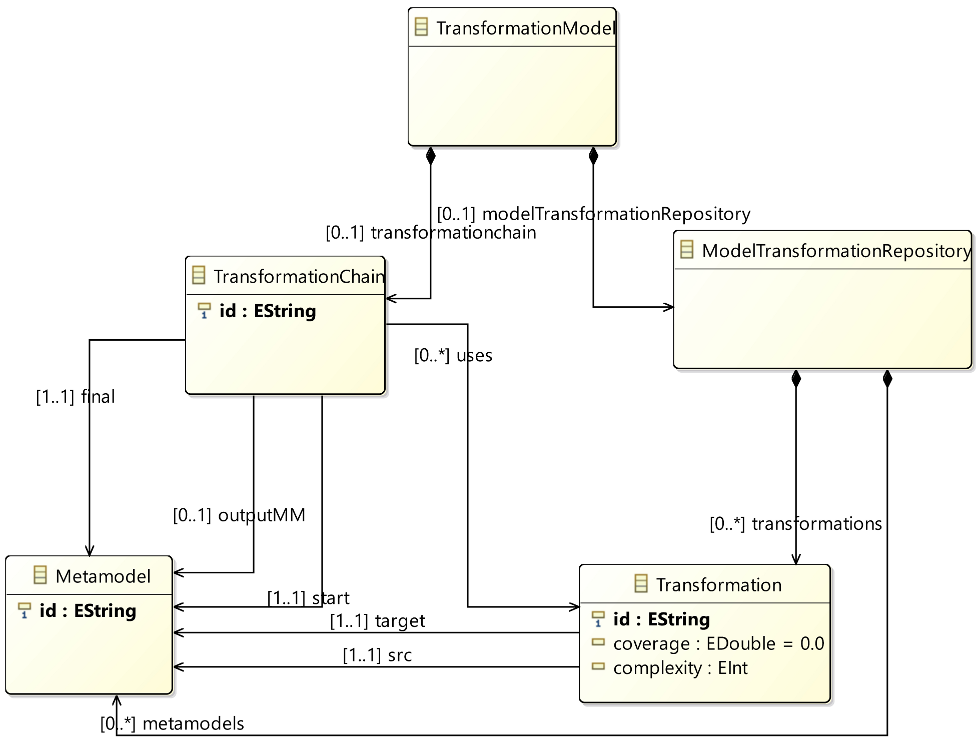


FIGURE 5.5: Metamodel for modeling the transformation chain selection problem

This metamodel allows to derive representations of the transformation chain selection problem, ones which serve as input to the MOMoT tool. This in turn leads to use of search strategies to achieve an optimal model evolution, and therefore, the involved transformation chain (TransformationChain). Models conforming to this metamodel are automatically created and persisted by the discoverer in ① after analyzing the given repository input. Indeed, this metamodel allows the creation of instances of TransformationModel in which ModelTransformationRepository and TransformationChain is contained. Consequently, ModelTransformationRepository is composed of Metamodels and Transformations. Each transformation has two references to src and target metamodel. Also, a TransformationChain can be referred to as an ordered set of transformations, that will be composed to satisfy the required input, i.e. the required start metamodel (derived by the given input model) and the required final metamodel indicating the targeted transformation output, with an additional (optional) outputMM that reflects the chain's current output metamodel. Consequently, for a non-empty chain, outputMM denotes the current output metamodel. It is required to realize the constraint that ensures feasible chaining solutions in that the chain leads to the output model as per user definition. Hence it is used by MOMoT in the search process to match it against the chains final instance. As long as the discoverer has persisted the *Repository model*, MOMoT can process it with the rest of the required artifacts.

Transformation Model for Chain Composition As part of the problem specification, transformation rules ③ facilitate the composition of a chain as an ordered sequence of (chain selection) transformations identified between source and the target metamodels. Note that the term "transformation" here is used ambiguously as it refers to 1) the mutation of the instance model that conforms to the domain model in Fig. 5.5, e.g., using Henshin units [15], which we use as part of the problem specification, and 2) the domain model element (which is an EClass) Transformation (cf. Fig. 5.5) where each instance implies a mapping between two metamodels, respectively. The former takes place with MOMoTs search engine applying rules to the model instance in order to explore its design space in terms of different chaining paths. The latter refers to possible hops within a solution to the chaining problem, which is reflected in the model by the Transformation instances referenced through uses relations from TransformationChain (*TC*). Altogether, MOMoTs solution to a task results from the changes, it imposes on the input problem model by means of rule transformations. Here, those reflect a selected Transformation instance to be executed on the model which we intend to derive a chain.

MOMoT uses Henshin [15], a graph-based transformation approach, to set the scope of possible changes for a domain model defined in EMF[3]. The tool-set includes a rule-based model transformation language to induce changes by exploiting a models graph representation, and a transformation engine to execute them. Using the concepts in our problem-describing model (Fig. 5.5), we can define patterns to match and imply changes in the model graph with so-called units and rules. Matches and the validity after change are hereby determined through formal reasoning.

Fig. 5.7 shows the Henshin transformation rules we defined to imitate the selection of a transformation *T* for a transformation chain *TC*. For the generation of a feasible chain, two cases can be distinguished in terms of matching semantics. Naturally, a *T* that is available in the repository, can be added to *TC* with rule addTransformation. Per rule definition, *T* is limited to candidates that guarantee executability of the resulting chain i.e., *T* needs to take as input that chains current outputMM. For the first application, however, the input of *T* needs to conform to the metamodel to be transformed, i.e., source of *T* corresponds to start of *TC*, as per user declaration. Taking the chaining example from Fig. 5.3, an excerpt of the instance model after selecting KM32EMF as first selected *T* is provided in Fig. 5.6.

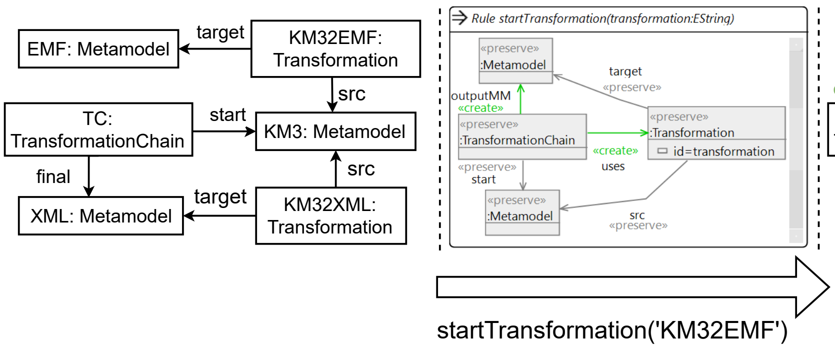


FIGURE 5.6: Example: Henshin rule for selecting transformation *KM32EMF* and resulting model instance (excerpt).

Following the rule application, the uses reference signifies KM32EMF to be part of the chain and EMF as thereafter emerging metamodel for the chain output, while metamodels defined essentially for the chains first and last *T*, start and final, are maintained. Consequently, rule startTransformation ensures that the first candidates' input corresponds to the language of the model to be transformed. In any case, the scope for applicable subsequent candidates is established with target of the appended *T* becoming outputMM of *TC*. The described conditional behavior is expressed with a ConditionalUnit, which applies one of the rules depending on whether a *T* is already part of the chain (rule checkHasTransformation). Alternatively, the same behaviour could be achieved with different control structures incorporated in other available Henshin units [14].

[3]https://www.eclipse.org/modeling/emf

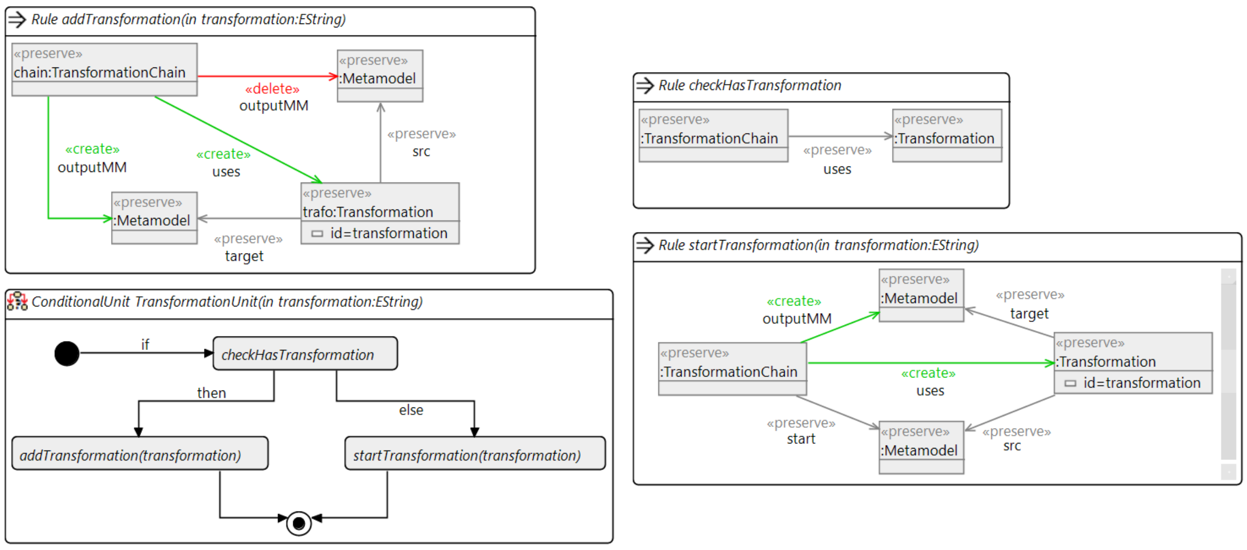

Rule addTransformation(in transformation:EString)
«preserve» chain:TransformationChain
«delete» outputMM
«preserve» :Metamodel
«preserve» src
«create» outputMM
«create» uses
«preserve» :Metamodel
«preserve» trafo:Transformation
id=transformation
«preserve» target
Rule checkHasTransformation
«preserve» :TransformationChain
«preserve» uses
«preserve» :Transformation
Rule startTransformation(in transformation:EString)
«create» outputMM
«preserve» :Metamodel
«preserve» target
«preserve» :TransformationChain
«create» uses
«preserve» :Transformation
id=transformation
«preserve» start
«preserve» :Metamodel
«preserve» src
ConditionalUnit TransformationUnit(in transformation:EString)
if
checkHasTransformation
then
else
addTransformation(transformation)
startTransformation(transformation)

FIGURE 5.7: Model transformation defining the chaining problem in Henshin

Defining Objectives and Constraints The *Quality Criteria* that can be used are various, but to demonstrate the approach we have chosen the ones anticipated in Section 5.2.1, i.e. transformation coverage, complexity and transformation hops, which are provided in the specification in ④. Two additional quality criteria are added as objectives that use to select the optimum transformation chain. They are metamodel similarity and model coverage. These objectives support chain selection from five different points of views/criteria, each of which adds a dimension to the fitness function that is used to evaluate derived chains.

We have used static analysis [106] of EOL and ETL (Epsilon Languages) to traverse through the elements of the used metamodels and model transformations respectively [5] and then calculate the three structural quality criteria/objectives for the transformation chains. The `transformation coverage` is determined through static analyses based on transformation and its involved target metamodel, i.e., transformationCoverage, and recorded in the intended attributes in the transformation afterwards. It is defined in coherence with the criteria discussed by Basciani et al. in [19]. According to them, the weight considered for a transformation rule covering a metaclass in input/output is unitary, whereas the individual binding of the transformation, predicating of a structural feature (in/out) weighs 0.5. Also, the `transformation complexity` is calculated by counting the number of constructs or elements used in a transformation module. These elements can be a predicate or reference to a metamodel element, or an EOL construct such as keywords or variables. Another simpler objective that define a transformation chain is the transformation `no. of hops` present in a transformation chain that transforms from one metamodel to another.

A user may define several other quality attribute that may be considered as a better heuristic to define a better transformation. The other two quality criteria or objectives that are considered in the experiment are `Model coverage` and `Metamodel similarity`. The `Model coverage` is the number of metaclass and structural features produced in the target model when compared to the target metamodel. It is considered to be inverse of the information loss concept that determine which modeling elements are considered in the targeted model [19]. This objective defines the exact modeling elements such as metaclass or structural features that are generated at the target model. Lastly, `Metamodel similarity` is done by transforming the metamodel into a graph using social network visualizer software[4]and then apply the graph similarity algorithm[5] to compare the

[4]`https://socnetv.org/`
[5]`https://github.com/wadsashika/Graph_Similarity_NM`

two metamodel used in the transformation. More the metamodel similarity, better the possibility of having easier transformation. It is a reasonable heuristic to define the transformation from one metamodel to another.

Any fitness functions can be defined using a configuration language provided by MOMoT, which is based on XBase [40], a Java-like, statically typed expression language. By using this language, the user can define quality criteria / objectives as part of the fitness function, which will be evaluated on the problem model to calculate transformation chain(s) optimally. The evaluation process was executed multiple times to estimate the mentioned indicators (such as Hypervolume and Generational distance) taken from MOEA framework based on multiple algorithms such as Random Search, NSGA-II, NSGA-III, etc.

Constraints in ⑤ enrich the specification to ensure the validity in TC i) in general and ii) with respect to the input model to transform. Accordingly, the output of the T having been most recently added to a chain (outputMM) poses a domain-specific constraint that limits the selection for a next T to those from the repository with a corresponding metamodel source (src). Moreover, the first Ts src needs to comply with the users input model to transform, which is delineated by start of a chain TC. Likewise, a valid chain ends with a T having the final output metamodel (target) corresponding to TCs final metamodel. Remarkably, our rule definition in Fig. 5.7 ensures that these constraints are satisfied for the chains that are eventually delivered in the result set.

Configuring the Search and Evaluation Next to the problem specification derived in previous paragraphs, the input to model-driven optimization tools like MOMoT usually entails configuring the search-based optimization process. Objective and constraint definitions elicit the extent to which a derived model reflects a desirable solution whereas for employed optimization techniques parameters and evaluation metrics have to be decided to support model mutation and facilitate collective, comparative quality assessment. Therefore, problem-related specifications were previously established upon the metamodel in Fig. 5.5, to define the search space, and declare the fitness and legality of derived transformation chains. The search space is designed in the problem model as shown in the Fig. 5.8. This problem model conforms to the problem metamodel. The experimental setup is now complemented with SBO-related (search-based optimization) settings (⑥) to facilitate reasonable exploration of chaining selection options. User-defined parameters thereby consist of the following: 1) Algorithms and associated parameter settings, 2) Termination criterion and runs, 3) Set evaluation measures and statistical test settings.

MOMoT allows to choose from a palette of generic multi-criteria approaches for targeted rule orchestration, ranging from evolutionary algorithms to local search to reinforcement learning techniques [27, 41]. In our evaluation, we let different algorithms compete against each other to demonstrate the algorithm-agnostic nature of this approach, and do so multiple times to compare their performance under statistical support on our selected example. The different algorithms we tried to select the optimal chain(s) as follows.

Random Search Random search algorithm uses some kind of randomness or probability in the definition of the method to solve an optimum problem. Typically, the basic random search algorithm is referred as simulated annealing. Some of the other random search are tabu search, genetic algorithms, evolutionary programming, particle swarm optimization, ant colony optimization, cross-entropy, stochastic approximation, multi-start and clustering algorithms, etc [124]. However, the random search used in momot framework randomly selects the best transformation as one of all possible solutions with equal probabilities of different objectives.

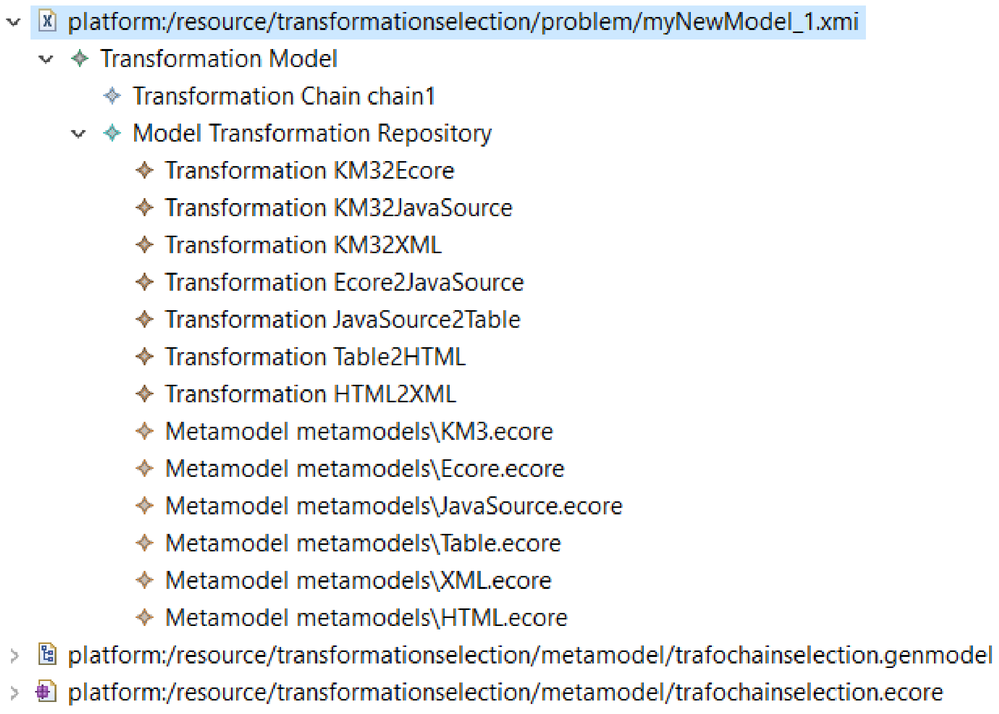

platform:/resource/transformationselection/problem/myNewModel_1.xmi
Transformation Model
Transformation Chain chain1
Model Transformation Repository
Transformation KM32Ecore
Transformation KM32JavaSource
Transformation KM32XML
Transformation Ecore2JavaSource
Transformation JavaSource2Table
Transformation Table2HTML
Transformation HTML2XML
Metamodel metamodels\KM3.ecore
Metamodel metamodels\Ecore.ecore
Metamodel metamodels\JavaSource.ecore
Metamodel metamodels\Table.ecore
Metamodel metamodels\XML.ecore
Metamodel metamodels\HTML.ecore
platform:/resource/transformationselection/metamodel/trafochainselection.genmodel
platform:/resource/transformationselection/metamodel/trafochainselection.ecore

FIGURE 5.8: Problem model to define search problem for MomoT

NSGA-II NSGA-II is a non-dominated genetic algorithm used for multi-objective optimization. It incorporates elitism and no sharing parameter needs to be extracted so that a diverse pareto-optimal set is sustained [36].

NSGA-III It is another non-dominated genetic algorithm which is a reference-point based multi-objectives that emphasized different population individuals which are not dominated yet close to a set of reference points [35].

EpsilonMOEA The epsilon MOEA algorithm is a steady-state algorithm which means only one member in the population is evolved per step. It uses epsilon-dominance in order to maintain a well-spread set of Pareto-optimal solutions [36].

Chebychev Chebychev algorithm uses a non-linear scalarized function which is used for action on different selected strategies in multi-objective reinforcement learning [113].

Pareto Q-learning A pareto Q-learning algorithm is a temporal difference learning algorithm which incorporates the relationship of pareto dominance into a reinforcement learning approach [114].

Hypervolume based learning A multi-objective reinforcement learning algorithm uses a hypervolume indicator as a strategy to action selection based on multiple assessment metrics from multi-objective optimization [112].

Tournament A tournament genetic algorithm is a modified NSGA-II that allows higher values of tournament size. It removes non-dominated sorting and replaced all children in the new population which further replaces the current population [66].

In general, the generated output contains feasible chaining solutions in terms of a pareto-optimal solution set, so the user has to reason about trade-offs in terms of quality criteria. Our goal is to find the best chain in terms of a singular quality or a set, to learn more about the trade-offs that apply to each of the several feasible transformation chains. Therefore, we run through a fixed number of evaluations in each run and with each algorithm. Nevertheless, the framework supports quality-based termination criteria when a particular fitness level is of interest for the desired solution. The search configuration we use is described in more detail in subsection 5.2.3.

Problem Encoding and Search Orchestration As mentioned earlier, MOMoTs optimization incentive is to determine rule sequences for the problem instance model to arrive at an objectively optimal model state with respect to the fitness function. The used Problem Encoding ⑦ is based on rules specifically designed to operate on model constituents of the particular domain. In our use case, each rule application represents a mapping between metamodels and acts as decision variable as part of a solution candidate for a chain TC. The effective arrangement of these applications is now the subject to the Search Orchestration ⑧. Naturally, how new sequences are generated depends on the used methods exploration and exploitation capabilities. A local search, for instance, is concerned with determining nearest neighbors by adding a transformation or replacing one in the current chain, to spawn new solutions. For the broadly adopted GAs (genetic algorithms), the framework initializes the population with legitimate chaining sequences at random. Moreover, the solution length in terms of the sequence of transformations (T) is limited to facilitate operators for alteration. In this respect, several operators for mutation and crossover are available. Note that transformation chaining poses a highly interdependent endeavour, thus chains emerging from recombination carry high potential of invalidity. For this reason repair mechanisms are foreseen to restore feasibility, e.g., by replacing non-executable transformations in the sequence.

As established earlier, a validity constraint ensures the chain ends at final output metamodel to conform to the problem specification. In fact, any TC concluding with a T having a target metamodel other than finalOutputMM represents an infeasible chaining path. These intermediary solutions however are maintained to be considered for further advancements, and meanwhile marked as invalid to later be omitted when deriving the final solution set.

Search Analyzer and Result Manager Through monitoring capabilities experimental setups in MOMoT are susceptible for clear-cut solution requirements and performance analysis. Information on the search process is collected and processed in the Search Analyzer ⑨ to enable premature termination settings, posing the option, e.g., to prioritize finding a chain that yields no attribute loss. When considering multiple optimization techniques, the MOEA framework is furthermore utilized to support performance evaluation. Chaining solutions evolved by different algorithms can be ranked using dedicated indicators like Hypervolume or Generational Distance. By default, they are computed post search with respect to the Pareto set holding the objective trade-offs. As a result, the best optimiser can be established for the chain selection task with support of statistical tests.

Upon search termination, the Result Manager ⑩ provides listings of the best found selection operations and therewith resulting output models, along with the optimal chains' (TC) quality criteria values. While the ordered Henshin unit instantiation leading to found chains and objectives are provided as textual output, the transformed model conform with the metamodel in Fig. 5.5 is persisted as (.xmi) model. Hence the chains transformation steps can be traversed programmatically and become subject to post-processing steps. This raises further options such as the immediate transformation of a model according to one of the found chains, visualization of found chaining paths with further details as shown in Fig. 5.9, and additional analysis effort. Indeed, we can extract the chains and depict them and add objective annotations for each mapping/translation step. This allows to identify bottlenecks in transformation quality, e.g., the mapping definitions responsible for most lost features/attributes.

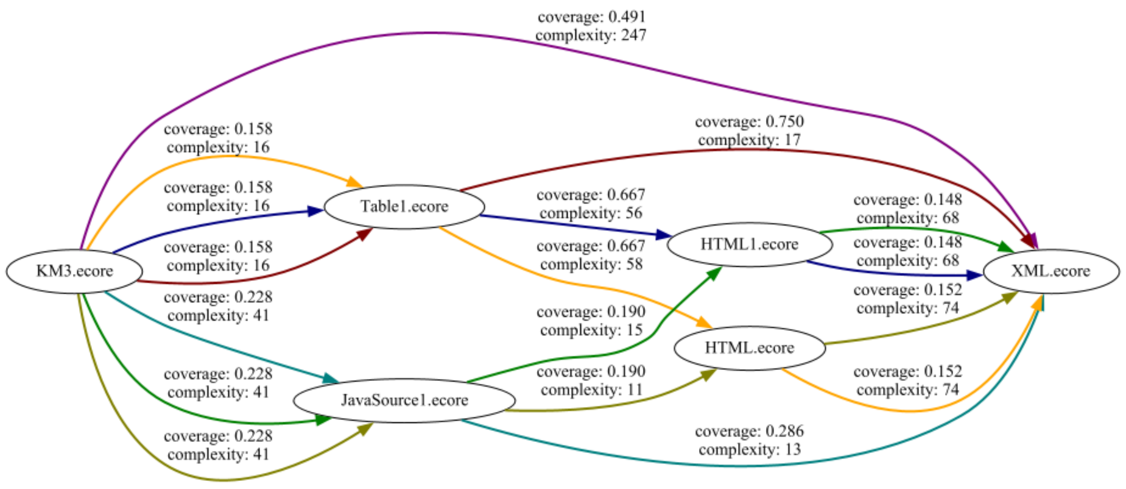


FIGURE 5.9: Generated visualization from identified chaining solutions

Result obtained in selecting optimal transformation chain Utilizing the described concepts in terms of the problem specification and with a search configuration available, MOMoT search engine can finally be employed to determine feasible chaining solutions as a matter of evolving the model instance. The output for the scenario described in Chapter 5.2.1 (c.f. Fig. 5.3) is shown in Table. 5.1. It includes the Pareto set of transformation chains produced by each algorithm and with respect to evaluated objectives, that is transformation coverage, complexity, model coverage, metamodel similarity and the transformation steps, i.e., hops. Note that for the transformation coverage, model coverage and the graph similarity are expressing a maximization objectives. Therefore, their additive inverse in objective functions is shown as negative (i.e. inverse of minimization). Moreover, the second available chain, Ch_1 with five number of hops is omitted from the output due to expressing a worse fitness in all conflicting quality respects. Apart from this, Ch_2 and Ch_3 have been determined by all eight algorithms and depict optimal solutions depending on the intended use. Accordingly, Ch_2 ($KM3 \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$) is lower in complexity than Ch_3 whereas Ch_3 ($KM3 \rightarrow XML$) has higher coverage and takes one transformation step only.

TABLE 5.1: Momot result of 8 algorithms to compute optimal chain(s) out of 3 identified chains

Chain	Tr. Coverage	Tr. Complexity	Model Coverage	MM similarity	nr. hops
Ch_2	-0.006610729	234	-0.4375	-0.0116795	4
Ch_3	-0.49122807	247	-0.4375	-0.1917	1

5.2.3 Experimental Evaluation

In this section we propose an evaluation of the approach based on two research questions:

RQ1: Is the approach able to retrieve the optimal chain based on the user-defined objectives: coverage criteria, complexity, model coverage, metamodel similarity and number of hops?

RQ2: How the performance of proposed approach is affected w.r.t. the size of the repository and input parameters based on different algorithms?

In the following we describe the experiment setup, discuss the results and threats to validity. All the experiments are run on a Windows 10 machine with 12 GB RAM that has i7-7500U CPU @ 2.70GHz-2.90 GHz. Some of the fixed search configuration are taken as follows. The population size of the experiment is taken as 6 and the maximum evaluation is taken as 12. In order to calculate execution time, we run the experiment 20 times for each of the eight considered algorithms. The entire corpus of etl transformations, models and metamodels is available on Github repository `https://github.com/lowcomote/chainselection_momot/tree/master`.

Experiment 1 In order to answer RQ1 we setup an experimental evaluation based on the ground truth established in [19]. In [19] the dataset used for the experimental evaluation is the same that we use in this paper. The graphical representation of the dataset is depicted in Fig. 5.3. We compare the results for best chains in [19] with the proposed approach. We have used the same coverage formalization, and since the transformations used in [19] are implemented with ATL, we have re-implemented the same transformations as ETL modules.

Results The results of the first experiment, used to answer RQ1, are reported in Table 5.2. For

TABLE 5.2: Results for RQ1

Chain	Tr. Coverage	Tr. Complexity	Model Coverage	MM similarity	nr. hops
Ch_1	0.0016	441	0.3125	0.0026	5
Ch_2	0.00661	234	0.4375	0.0116	4
Ch_3	0.49123	247	0.4375	0.1917	1

each chain identified by the approach, the best chain, selected based on the maximum transformation coverage value, is chain Ch_3 with 0.491. Meanwhile chain Ch_2 is a better chain in terms of transformation complexity. But chain Ch_1 is the worst chain in terms of all the objectives when compared to the other chains.

Considering transformation coverage and model coverage (inverse of information loss) only, the result confirms that the selected optimal chain would be Ch_2 and Ch_3 as in the ground truth in [19].

By also considering complexity and number of hops, the approach would consider minimum complexity with the minimum number of hops. According to Table 5.2, chain Ch_2 has minimum complexity (234) and chain Ch_3 has the minimum number of transformation hops, i.e. 1. Since chain Ch_3 has the highest coverage, metamodel similarity and minimum number of hops, we can assume it to be the optimum transformation chain available, considering the weights of the two quality (objective) criteria in our approach as the same. The weights are given by the user based on chain quality requirements. For simplicity, we consider them equal. Although there is no ground truth upon which metamodel similarity can be tested, based on the transformation coverage, we can claim that Ch_2 and Ch_3 are the optimal transformation chain.

Experiment 2 In order to answer RQ2, we have executed another experiment with different parameters having different algorithms to run the MOMoT experiment in the defined transformation chain repository. The different parameter configurations are shown in the Table 5.3 and Table 5.4. The defining objectives are transformation coverage, transformation complexity, model coverage, metamodel similarity and number of hops. We have taken two problem models so as to experiment with different sizes of model repositories upon which different search algorithms would perform under the prescribed parameters configurations. The first problem model's specification is as follows.

TABLE 5.3: Different parameters configuration_1

Paramters	Population size	Max evaluation	nr. runs
Parameter1	10	300	5
Parameter2	20	600	10
Parameter3	40	1000	30
Parameter4	70	1400	50
Parameter5	100	2000	70
Parameter6	120	3000	80

TABLE 5.4: Different parameters configuration_2

Paramters	Population size	Max evaluation
Parameter1	10	300
Parameter2	20	600
Parameter3	40	1000
Parameter4	70	1400
Parameter5	100	2000
Parameter6	120	3000

Problem model 1 The first defined problem model is having 11 metamodels, 40 ETL transformations and 97 possible transformation chains between the user-defined source and the target metamodel. The source and target metamodels are KM3 and XML respectively. The average execution time taken to get the optimum result according to the various algorithms based on different parameter configurations are shown in Table 5.5 and Table 5.6. In these two tables, the parameters (abbreviated as **P**) are shown in the first column as discussed in Fig. 5.10 and Fig. 5.11 respectively.

TABLE 5.5: Average execution time (ms) for various algorithms for configuration_1

P	Random	NSGA-II	NSGA-III	$\varepsilon -$ MOEA	Chebycheff	PQ-HV	PQ-PO	Tournament
P1	2215.8	910.4	2040.6	1177	1883	2076.6	1734.2	1693.2
P2	4933.7	1905.9	1508.9	2598.3	2728.4	2346.5	2252.9	2290.6
P3	8158.03	3295.13	3374.4	3731.1	3464.23	4467.43	3993.76	3801.7
P4	14256.04	7147.82	4920.54	7015.62	6824.72	6933.56	6623	6484.3
P5	20992.48	9629.98	9240.27	13613.73	8559.1	10844.38	10918.74	10910.87
P6	31523.17	12544.3	11948.7	14693.24	14566.25	15464.88	16419.04	13249.52

TABLE 5.6: Average execution time (ms) for various algorithms in configuration_2

P	Random	NSGA-II	NSGA-III	$\varepsilon -$ MOEA	Chebycheff	PQ-HV	PQ-PO	Tournament
P1	2424.8	1374.2	864.65	1009.95	976.55	1700.2	1122.85	1319.6
P2	4224.65	2266.9	1580.95	2424.8	2367.35	2469.35	2581	2507.5
P3	7991.5	3186.9	3284.8	3286.85	3383.2	3773.05	3577.75	3387.25
P4	10371.05	4701	4084.9	5347.4	4187.6	5254.2	5377.65	4611.1
P5	15920.9	6805.8	6351.85	6777	6604.55	7161.65	6904.2	6583.45
P6	22264.85	9348.3	7731.95	10449.6	10082.55	12540.55	11639.45	10557.6

The MOMoT framework is executed on the problem model of the chain selection use case and the execution time of different algorithms based on different parameters is highlighted in Fig. 5.10. This figure shows that the Random search algorithm is the slowest whereas the NSGA-III is the fastest search optimization algorithm.

Problem model 2 The first defined problem model is having 6 metamodel, 12 ETL transformations and 13 possible transformation chains between the user-defined source and the target

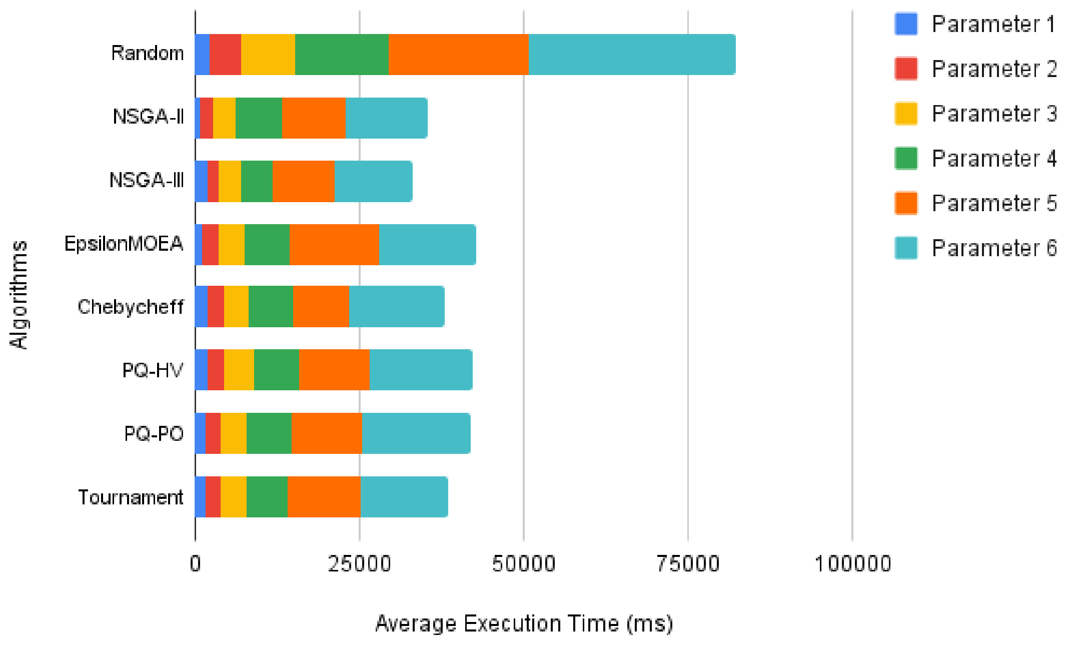


FIGURE 5.10: Execution time of different algorithms w.r.t parameter configuration_1 in problem model 1

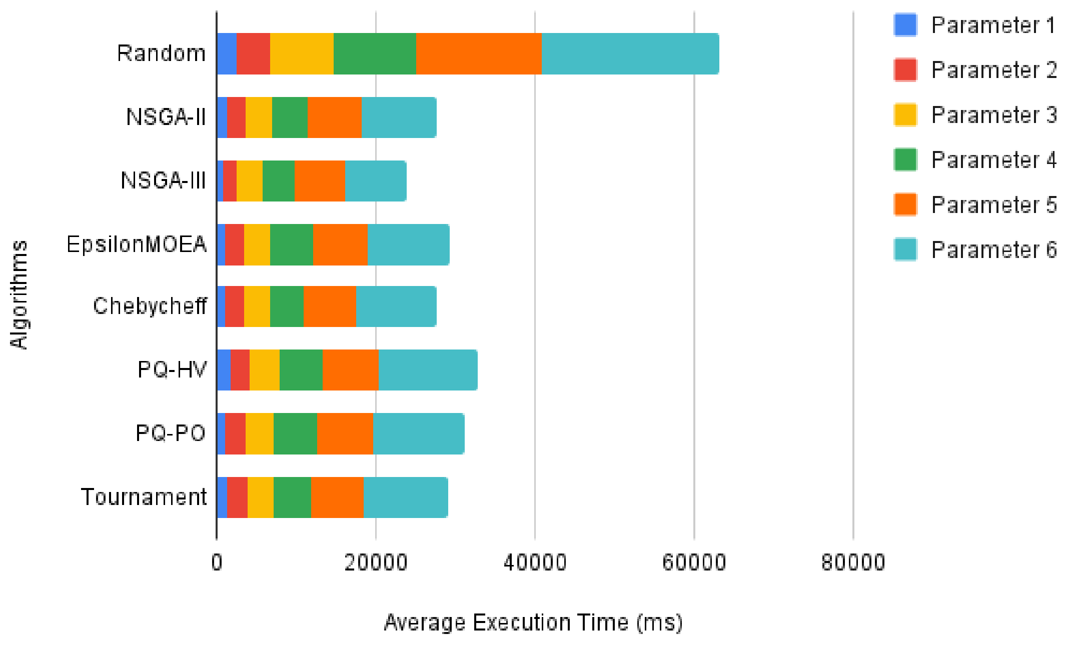


FIGURE 5.11: Execution time of different algorithms w.r.t different parameter configuration_2 in problem model 1

metamodels. The source and target metamodels are KM3 and XML, respectively. The average execution time taken to get the optimum result according to the various algorithms based on different parameter configurations are shown in Table 5.7 and Table 5.8.

The MOMoT framework is executed on the problem model of the chain selection use case and their execution time of different algorithm based on different parameter configurations are highlighted

TABLE 5.7: Average execution time (ms) for various algorithms in configuration_1

P	Random	NSGA-II	NSGA-III	$\varepsilon - $MOEA	Chebycheff	PQ-HV	PQ-PO	Tournament
P1	4225.4	921.4	2065.6	885.8	1532.9	1723.7	1659.2	2075.9
P2	4933.7	1905.9	1508.9	2598.3	2728.4	2346.5	2252.9	2290.6
P3	7534.67	2522.67	1830.6	2358.93	2583.07	2693.87	2863.77	2641.7
P4	10264.84	3723.6	2941.02	2851.86	3582.68	3936.96	3971.5	4222.92
P5	15283.87	4623.06	4987.01	4593.36	4951.41	5762.83	5754.14	5470.73
P6	25681.46	7205.56	6683.97	5872.17	7143.61	8346.07	8234.62	7421.37

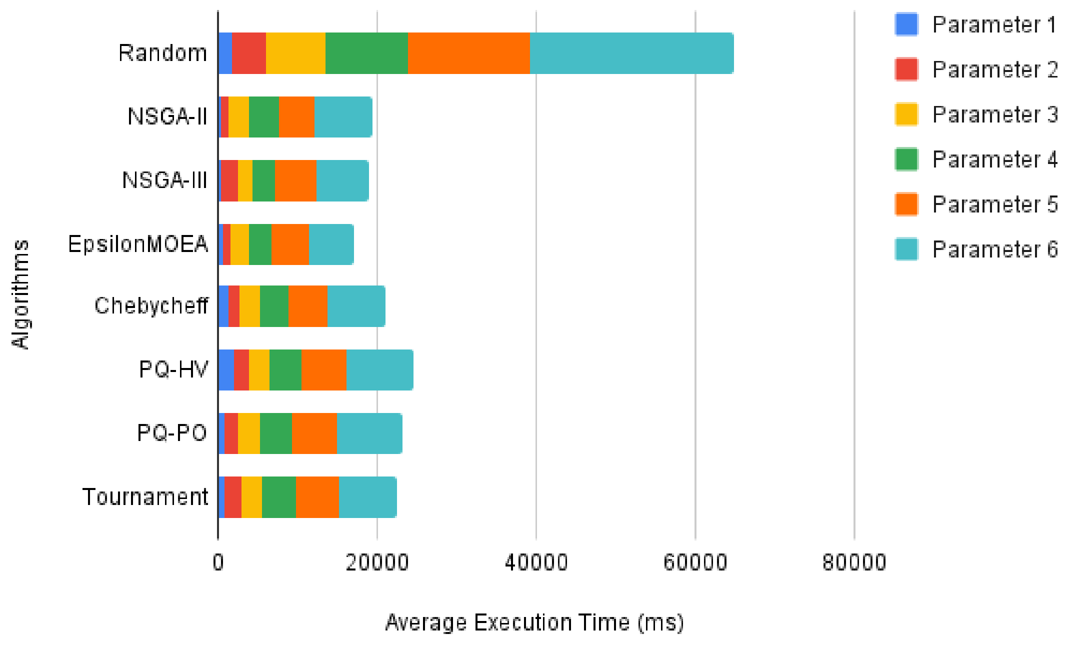


FIGURE 5.12: Execution time of different algorithms w.r.t different parameter configuration_1 in problem model 2

TABLE 5.8: Average execution time (ms) for various algorithms in configuration_2

P	Random	NSGA-II	NSGA-III	$\varepsilon - $MOEA	Chebycheff	PQ-HV	PQ-PO	Tournament
P1	4541.1	1783.7	1200.6	989.8	714.3	1523.2	1504.1	837.3
P2	4496.35	1566.1	1932.6	1718.75	1335.95	1753.95	1632.4	1476
P3	6786.05	2941.3	2509.05	2504.3	2391.3	2711.65	2423.7	2217.4
P4	10633.65	4086.95	3953.45	2925.95	3377.9	3537.5	3342.5	3336.8
P5	15045.8	5842.15	6857.6	4165.8	4300.6	5494.2	5060.55	4058.55
P6	23126.1	7933.4	9205.4	8095.2	7045.3	7662.5	7631.4	7674.25

in the Fig. 5.12 and Fig. 5.13 respectively. This figure shows that the Random search algorithm is the slowest whereas the Epsilon MOEA, Chebycheff and Tournament algorithms is the some of the fastest search optimization algorithm when it comes to consider lesser number of transformation chains.

Discussing problem model 1 and problem model 2 Both the execution time versus algorithm shown in Fig. 5.10 and Fig. 5.12 for problem model 1 and 2 respectively show that the random search is the slowest algorithm to compute the optimal chain used in MOMoT framework. But the difference between the two figures is that in the larger problem model 1, the genetic algorithms NSGA-II and NSGA-III are the fastest algorithm and EpsilonMOEA is the second slowest

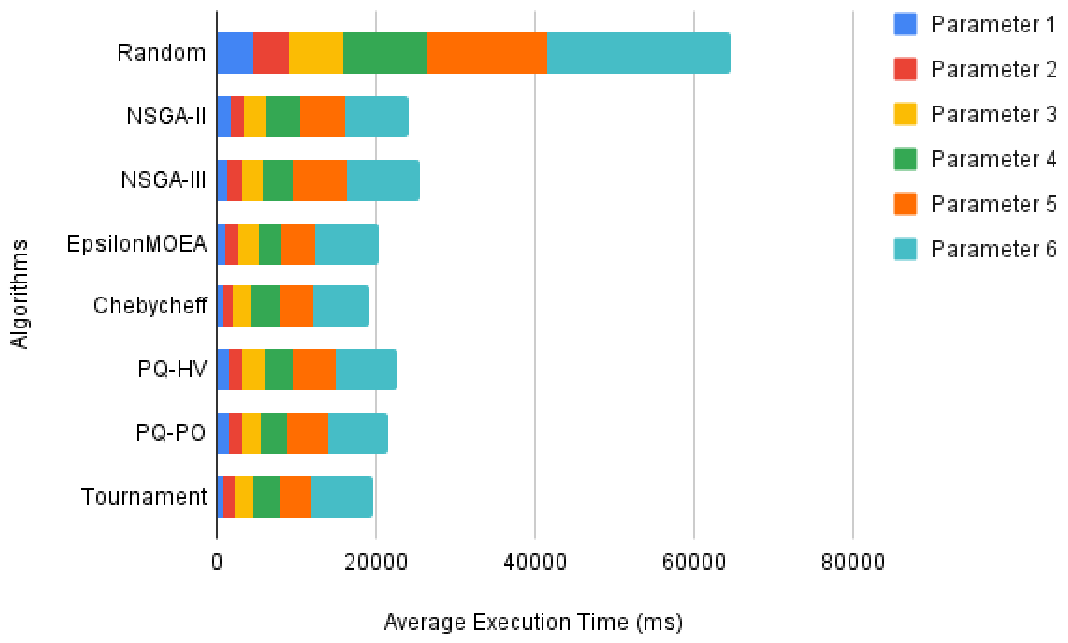


FIGURE 5.13: Execution time of different algorithms w.r.t different parameter configuration_2 in problem model 2

after random search whereas in the smaller problem model 2, EpsilonMOEA, Chebycheff and Tournament algorithms work better in calculating the optimal transformation chain(s).

5.2.4 Threats to Validity

In this section, we discuss the threats to validity of our experiments, by distinguishing them in internal and external.

Internal Validity Such threats are the factors that could have influenced the final results of the performed experiments. We attempted to avoid any bias in the definition of the quality criteria we have used, since they can influence the chain selection. To mitigate such threats, we have considered the same coverage formalization as in the ground truth paper [18, 19]. Moreover, we have built our dataset, by translating the existing transformations in the dataset in [19] from ATL to ETL. This can include different transformation constructs used by the two transformation languages, e.g., operations in ETL vs. helpers in ATL. We think that using only declarative aspects of both the transformation languages would limit this threat of varying quality criteria between ATL and ETL files. However, we have written almost syntactical equivalent ATL constructs into the ETL file.

We are aware that we have used a relatively small dataset of transformations, but it is based on real transformations with very different complexity, definitions, constructs and domain of applications. We have also tried to mitigate this aspect by applying a mutation of the dataset, which is the result of changing the required input and artifacts included in the repository.

External Validity The external validity discusses whether we can generalize our results. The first aspect we need to highlight is that the experiment has been based on an existing dataset of transformations, which has been rebuilt starting from a snapshot of the ATL transformation zoo [16]. The second aspect we need to discuss is that our approach is based on ETL transformations static analysis, but the approach is completely generalizable to other transformation languages on which quality criteria can be defined. To confirm that in the first experiment, we have compared the coverage-based selection with the paper in [19], where the transformations were defined with ATL. We mitigated this aspect also having modularized the static analyzer of the transformation that can be replaced with another one, in order to create the repository model. Moreover, in our approach, the static analysis [106] operates with a pre-requisite that the ETL module must contain the source and the target metamodels' linked in the header of the transformation. This avoids extra operations to retrieve the graph of the possible chains. We think that this threat is not influencing the results of the experiments since by using the approaches presented in [37, 38] the recovery of the existing chains is possible with multiple technologies.

5.2.5 Discussion

Search-based optimization techniques for selecting the optimum transformation chain involve using search algorithms, such as genetic algorithms or simulated annealing, to explore the space of possible transformation chains. The objective is to find a chain that optimizes certain metrics, such as transformation coverage, complexity, graph similarity, and number of hops.

The transformation coverage objective aims to find a chain that covers as much of the input space as possible. Complexity objective aims to find a chain that has the lowest complexity. Graph similarity objective aims to find a chain that preserves the structure of the input as much as possible. The number of hops objective aims to find a chain that has the least number of transformations.

It is important to note that these objectives may be conflicting, and trade-offs may need to be made. For example, a chain that has high transformation coverage may also have high complexity. Therefore, a multi-objective optimization approach may be required to balance these conflicting objectives. This is the advantages of using a multi-objective optimization algorithms.

In the search-based optimization technique, the algorithm starts with a randomly generated set of transformation chains, which are then evaluated based on the objectives. The best performing chains are then used to create new chains through genetic operations such as crossover and mutation. This process is repeated until a satisfactory solution is found or a stopping criterion is met.

Overall, search-based optimization techniques for selecting the optimum transformation chain are a powerful and flexible approach that can be used to find good solutions in complex and large search spaces. However, it is important to ensure that the objectives are well-defined and the search space is properly defined and understood.

5.3 Summary

Search-based optimization techniques are crucial in selecting the best transformation chain(s) based on multiple objectives in model transformation. These objectives include transformation coverage, transformation complexities, graph similarities, and the number of transformation hops. Search algorithms like genetic algorithms, simulated annealing, and other optimization algorithms are employed to achieve this. These techniques explore the solution space, enabling the identification of the transformation chain that best satisfies the desired objectives. The transformation coverage objective ensures that all elements in the source model are properly transformed to the target model, while the transformation complexities objective aims to minimize the complexity of each transformation step. Additionally, the metamodel similarities objective measures the similarity between the source and target metamodels, and the number of transformation hops objective seeks to minimize the number of steps needed to reach the final solution. By leveraging search-based optimization techniques, developers can effectively identify the optimal transformation chain that aligns with their specific user objectives.

Moving forward from the identification and selection of the optimal transformation chain(s), the next step involves optimizing the execution of these chains to improve the overall execution time. This optimization process is addressed in the next chapter, which focuses on reducing unnecessary modeling elements and transformation rules that do not contribute to creating the desired output model. By streamlining the transformation process, this optimization effort aims to decrease the execution time of the model transformation chains, ultimately enhancing efficiency. Furthermore, reducing generated target elements produced during the transformations within a specific transformation chain contributes to a more concise and targeted output model. Through these optimization techniques, developers can significantly improve the performance and effectiveness of the transformation process, enabling faster and more streamlined model transformations.

Chapter 6

Optimizing the execution of model transformation chains

Although in model transformation chains, the individual model transformations might be simple or complex, the entire chain is a combination of all the transformation constructs referring to metamodel elements that are used to propagate the information contained in the models. The transformation coverage [19] is defined as a quality characteristic defining how much the transformation "covers" the source and target metamodels' concepts. The result of this considered quality attribute is that every transformation involved in the chain can consider all the metamodel concepts of all the involved metamodels in a transformation chain. This results in a waste of resources and time in the execution phase, while in a transformation chaining process, all the transformation rules will be tried to match, even if the meta-elements are not propagated during the chain.

In this chapter, we propose an approach for optimising model transformation chains that leverage static analysis of the transformation and performs rule analysis to execute automated program rewriting behind the scenes. The main idea is to remove those transformation rules and bindings which generate those intermediate model elements that are not propagated through the final target model.

6.1 Background and Motivating Example

In this section, we introduce the tools and the languages used for our proposed approach. Model transformations are seen as the heart and soul of MDE [103]. Model-to-model (M2M) transformations transform a source model into a target model in the same or different abstraction level (metamodel), whereas a model-to-text transform models into source code or other text. In this section, we concentrate on M2M transformations only through which we can consider optimizing the metaclasses and structural features elements of the involved metamodels and transformations through the chain.

The Epsilon Transformation Language (ETL) [1], is the M2M transformation language provided by the Epsilon family [64]. ETL takes a particular source modelling elements from the source metamodel and transforms them to a number of target modelling elements from the target metamodel. This means that an ETL *module* can contain a number of transformation rules which transform a source model element to one or more target model elements. An ETL module can also have pre and post block to be executed before and after the execution of transformation rules respectively. Every rule is composed of an internal body in which the user can specify the *bindings*. A binding is used to set the features of an instance transformed by the current rule, by using the values of the features of the source instance. This aspect is mostly supported by the declarative nature of the transformation language, even though the user can also use imperative constructs as ETL is a hybrid M2M transformation language. In order to support reuse and maintainability,

[1]https://www.eclipse.org/epsilon/doc/etl/

complex model transformations can be composed through small transformation modules. The composition can be external or internal. External composition deals with composing model transformations together by passing models from one transformation to another. Internal composition composes two model transformations definitions into one new model transformation, expressed in the same transformation language[43, 118]. The external composition can be enabled if some pre-requirements are respected, e.g., the target metamodel of the first transformation T_1 is used as the source metamodel of the next transformation T_2. For our case, we are considering only external composition.

In the transformation chaining scenario represented in Fig. 6.1, we need to chain the two existing transformations, i.e. *A2B* and *B2C* satisfied, giving place to a chain $C_1 = A2B \rightarrow B2C$.

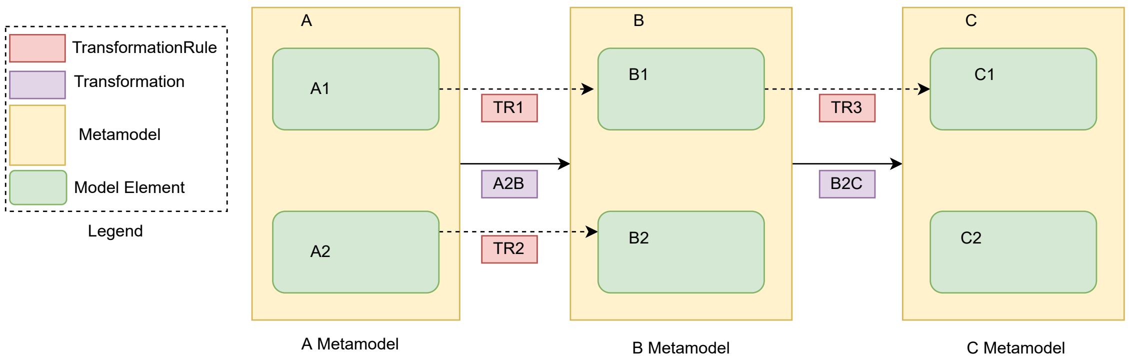


FIGURE 6.1: An example model transformation chain

Transformation *A2B* is composed of two transformation rules *TR1* and *TR2*. *TR1* transforms metaclass *A1* instances into *B1* instances, while *TR2* transforms *A2* instances to *B2* instances. Transformation *B2C* has one transformation rule *TR3* that transforms *B1* model elements to *C1* model elements. Now if analyze chain C_1 from start to end, we can notice that rule *TR2* is generating *B2* model elements which are not propagated in the final transformation *B2C* since there are no specific rules matching B2 elements.

In this case, the available chain is only one, i.e. C_1; hence given a model conforming to metamodel A, the chain can be executed to get as output a model conforming to metamodel C. If we execute the chain by composing two transformations, all the rules and bindings in the transformations will be executed, by trying to match all the elements declared in the transformations chain, even if they are not propagated by the intermediate or following transformation. For instance, in this case, the first transformation will execute rule TR2, producing elements, even if in the following transformation TR3, elements of type B2, are not considered nor propagated. This might cost unnecessary overheads in terms of execution time, especially for larger chains. This case is quite trivial for the sake of simplicity, but complex transformations, in which multiple rules are declared, and multiple transformations are used, can require optimized composition. This lead to the need for an optimization phase that can be performed before executing the chain, which we propose in the following section 6.2.

The motivating example in showing the optimization of a chain is shown in Fig. 6.2. In this figure, two transformations were chained in which only those statements in the first transformation is executed which is considered as a source element in the second transformation. For example, statement $s22 = s12$ is not executed in *A2B* transformation, because s22 is not considered as a source element in the transformation *B2C*. If no statement in a transformation rule is executed, then the whole rule is removed from the optimized version of the transformation.

In order to tackle statements with equivalent operators, a dependency graph is built that defines dependencies between two transformation rules based on a given equivalent(s) operator. For

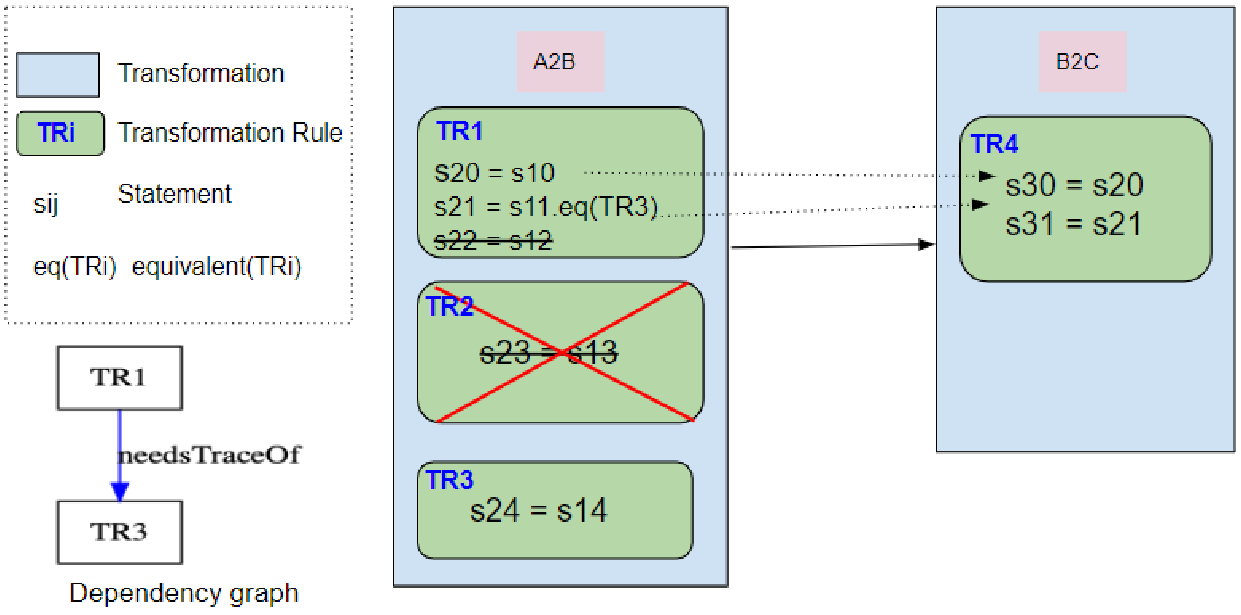


FIGURE 6.2: An example optimizing model transformation chain

example, statement *s21 = s11.eq(TR3)* in rule TR1 will cause dependency to rule TR3. So, if this statement in TR1 is executed, then automatically the statement with the same class type in TR3 is also executed.

6.2 Proposed Approach

In this section we propose an extension of the transformation architecture proposed in [19], that is able to optimize the selected chain of model transformation(s), before executing it. The expected benefits for this application are expected to be in terms of a reduction in memory allocation and execution time.

The architecture proposed in [19] is composed of the components in gray whereas the yellow one has been added to support the optimization (shown in Figure 6.3). The user input is composed of three elements: 1) a model repository (that might be also a local folder) containing modeling artifacts, 2) an input model and metamodel, and a 3) required output metamodel.

The *Chain Discoverer*① is the component that given the user input, explores the graph-based representation of the repository looking for available transformation chains satisfying the user request. Alternatively, we can implement an algorithm on a repository/folder in Java. Identification of possible model transformation chains can be done using following steps. The first step is to check if the direct model transformation exists that transforms source model to target model. This is the base condition. Then we identify all the available model transformations that transform source model to any of the available intermediate models. Further, we reuse the intermediate model as the source model with the help of base condition (using depth-first search recursively) and we repeat this step till we reach to the target model. Then, we check all the available model transformations that transform the identified intermediate model to the target model. Lastly, the source, intermediate and target models are composed and returned as a model transformation chain. The explained algorithm for identifying the possible transformation chain between a source and target metamodel is shown in the Algorithm 1 and it is explained in Chapter 5.1.

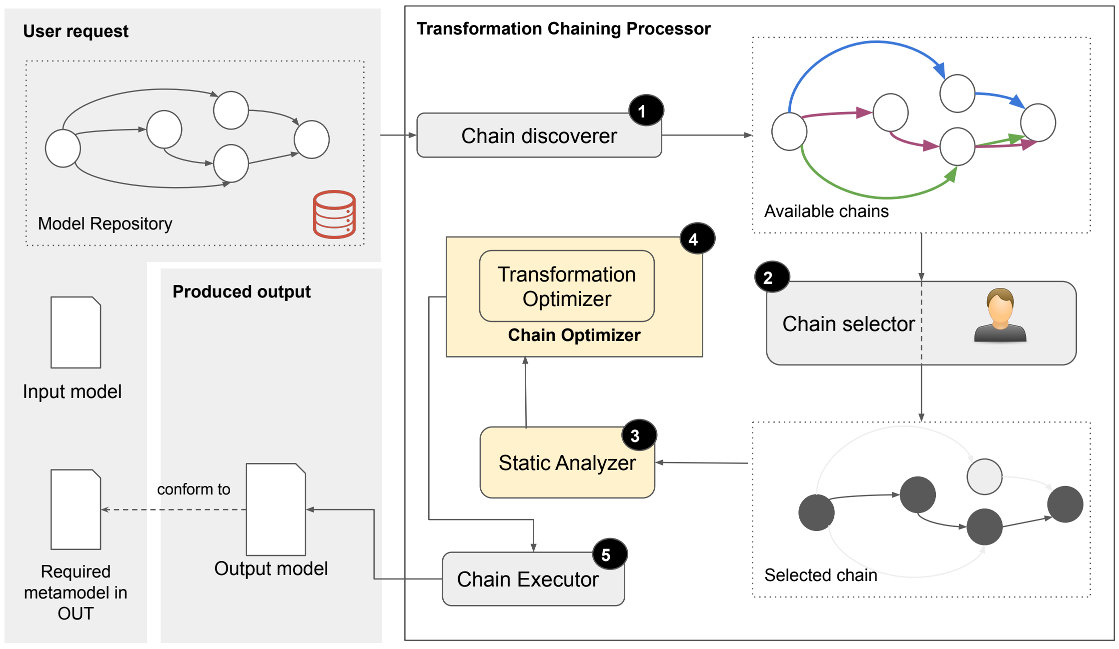


FIGURE 6.3: Proposed approach optimizing the execution of transformation chain

If multiple chains are available, the *Chain Selector* selects one or more optimal chains, by using user preferences, criteria or directly the user selection. We have selected the optimal transformation chain(s) by using MOMoT framework [44] and uses multiple criteria such as transformation coverage, complexity, model coverage, metamodel similarity and no. of transformation hops as discussed in Chapter 5.2 [95]. MOMoT uses different search-based optimization algorithm such as NSGA-II, NSGA-III, epsilon-MOEA, etc. to find the optimized transformation chain. If only one chain is available for that request, the *Chain selector* ② is skipped. When the approach has the selected chain, it can be executed to return the model in output, but a new component is then invoked, i.e. the *Chain Optimizer*④, that is in charge of optimizing the selected chain using the *Static Analyser*③ and pass it to the *Chain Executor*⑤ that will run the chain and save the output model.

In the following we detail the optimization phase and we explore the components used for this purpose. The optimization that we propose executes only those transformation constructs that are needed, only propagating intermediate model elements which are required to generate target model elements. This concept is illustrated in Figure 6.1, where we demonstrate that the rule TR2 is not needed since the chain at the second step does not consider the target metaclass of that rule. This concept can be easily extended to internal bindings of the rules which is explained in Figure 6.2. The required rules are derived based on the extracted typed information from the *Static analyzer* component③. The main idea is to analyze the transformation and then the entire chain from the initial source to the final target. The static analyzer is a helper module based on Java and Epsilon, which allows interacting with a model transformation as a model, so it can be queried and managed for instance by using EOL scripts [4]. For every transformation rule, we check whether the target parameter(s) of the source model is the source parameter of any rule in the next intermediate or target model. Only the matched transformation rule is chosen to be the required rule. Otherwise, that transformation rule is removed from the transformation, to speed up its execution. We can extend this logic to every statement of the transformation rule in which the transformed references and attributes of a particular metaclass are checked for being present in the next transformation, i.e. within a binding of a rule in that transformation. Alternatively, if the transformed element is not present in the next transformation then that particular statement of the rule is deleted. It is important to preserve the semantics of the transformations during this optimization process. We ensure that using the equivalence testing. We execute both the optimized and original transformations and then compare the output models.

The static analyzer first invokes the algorithm presented in [4], in charge of building the dependency graph between the rules based on the equivalent(s) operator used in statements of those rules [111]. The equivalent operator is a built-in operator of ETL which automatically resolves features of source elements to their transformed counterparts in the target models. The output of this algorithm gives a hashmap in which the *key* contains all the rules in a given transformation while the *values* contain the dependent rule(s) of the corresponding keys(rules).

The hashmap output of this dependency graph algorithm is the input for the Algorithm 2 that traverses each statement of the given rule in the transformation file. This algorithm 2 checks the target bindings of a statement in a transformation and compares them with the source bindings of the next transformation. If the target binding (in a transformation) matches with the source binding (in the next transformation), we store it in the `RulesToKeep` array, otherwise, we store it in the `RulesToDelete` array. Algorithm 2 then checks the *values* given in the `HashMap` and compares them with the values stored in `RulesToDelete` array. If they are equal, then the reference of the current rules from `RulesToDelete` array will be removed from it. Otherwise, the current rules are not being dependent on any other rule; therefore, such rule can be a part of `RulesToDelete` array so that it can later be deleted to optimize the overall transformation chain.

One challenge here is that rules can potentially alter model elements. The use of the "equivalent" expression in ETL can only be used to find transformation rules that depend on other rules of a model transformation. To handle this issue, we propose constructing a dependency graph for finding such rules and to perform the program rewriting after analyzing this dependency graph. This dependency graph is described in the paper [111].

The logic of the optimization stage is shown in Algorithm 2. This algorithm checks hither a model element feature (attribute or reference) in a particular input pattern or binding of a transformation rule of the current transformation is used in any transformation rule of the next transformation. Once the chain is optimized, we calculate the number of bindings in the rules of the optimized transformation. If there are no bindings in the rules of an optimized transformation, then that rule is deleted from the optimized transformation. If the binding in the current transformation is not used in the next transformation, then this binding will be deleted as it is unused for that specific next transformation. Once all the unused bindings from transformations are removed, we have the optimized transformation chain which will be executed to retrieve the same result with lesser generated elements and execution time.

Finally, the rewritten transformations containing only the required transformation rules are executed within a chain of model transformation. The Algorithm 2 is written in Java language and its code is defined in the github[2] repository [3].

6.3 Experimental Evaluation

In this section we evaluate the approach by answering to the following research questions:

RQ1: Is the approach able to produce correct results w.r.t. non-optimized chains?

RQ2: Is the approach effective in optimizing the execution time of the available chains varying model size or transformation chain hops?

[2]`https://github.com/lowcomote/chain-optimisation.git`

Algorithm 2 Pseudocode for the optimization of transformation chains

```
 1: procedure OPTIMIZE(HashMap hm)
Require: DG = Dependency Graph given in FindDependencyGraph(a)
 2:     Let a = chain containing transformations a0, a1, ..., an
 3:     Traverse each transformation in the given chain a
 4:     Take consecutive pair of the transformations from end to start of the chain (an, an − 1)
 5:     Source = an − 1
 6:     Target = an
 7:     BindingsToDelete, BindingsToKeep        ▷ Arrays for keeping the bindings that need to be
    removed and the bindings to keep, respectively
 8:     for all rule=rules in Source do
 9:         for all binding=bindings in rule do
10:             ref_type = EReferenceType of binding
11:             TP = types of target parameters of the binding in the rule in transformation a(i − 1)
12:             SPs = types of source parameters of the binding in the rule in the transformation ai
13:             dependent_rule = values given in key rule stored in hashmap hm
14:             if SPs equals TP then
15:                 add binding in BindingsToKeep
16:             else
17:                 if ref_type = dependent_rule then
18:                     add binding in BindingsToKeep
19:                 else
20:                     add binding in BindingsToDelete
21:                     remove binding from Source
22:                 end if
23:             end if
24:         end for
25:         if (#binding in the rule = 0) then
26:             delete rule
27:         end if
28:     end for
29: end procedure
```

RQ3: What is the relationship between generated target elements and execution time of the optimized and unoptimized chain?

6.3.1 Experiment setup

The experiment is based on a case study borrowed from [19] and reported in the dataset online `https://github.com/lowcomote/chain-optimisation.git` written in the ETL transformation language. The case study is composed of 6 metamodels and 6 transformations. The user request is made by giving an input KM3 model and requires an XML model as output; the repository is represented in Fig. 6.4. The experiments are run on a Windows 10 machine with 12 GB RAM that has i7-7500U CPU @ 2.70GHz-2.90 GHz.

Experiments done We have executed the approach on this case study by providing the user request, i.e. the initial model and required metamodel, as well as the repository of the case study. We executed the available chains 10 times and get measured the average of those runs. We reported all the execution times and calculated the entire chain execution time with/without the optimization component. If more than one chain is available for the case study, we calculated all the possible

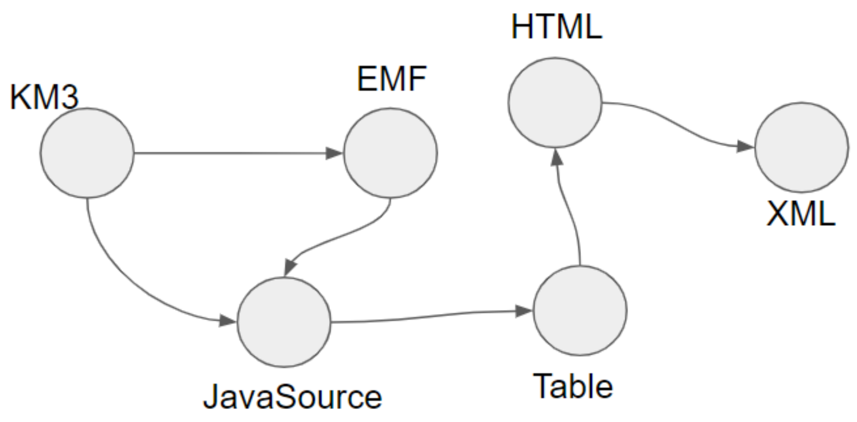


FIGURE 6.4: Graph based representation of the KM32XML experiment

chains for both versions, optimized and unoptimized. If the results showed an improvement in terms of execution time, we compare the resulting models produced by two executions in order to check if the models are exactly the same. This would confirm that the approach is able to optimize the execution based only on the removal of unneeded modelling constructs of the transformation. This demonstrates the correctness of the implemented optimization algorithm. In order to compare the output models we have used EMFCompare [1] to automate the tasks and we inspected the results to confirm that the two models are exactly the same or not. If the models are the same, we can check the execution time, otherwise, we mark as unexpected result of the optimized transformation chain. Also, the cache memory to load the EMF models has not been considered in order to exclude possible wrong evaluations based on caching features. The models used for the experiments are an original model provided by the case study and a set of randomly generated models. The generated models are obtained by using EMG [3], which is the Epsilon Model Generation language supporting the semi-automated generation of models [84]. With this tool, it is possible to create user-defined instances of metaclasses in the metamodel(s), assign values to the instance's attributes, and create links between instances to assign values to references. We generate 16 models for each case study to have in total 32 models to test the approach, having different sized models for the two transformation chains as shown below.

CH_1: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

CH_2: $KM3 \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

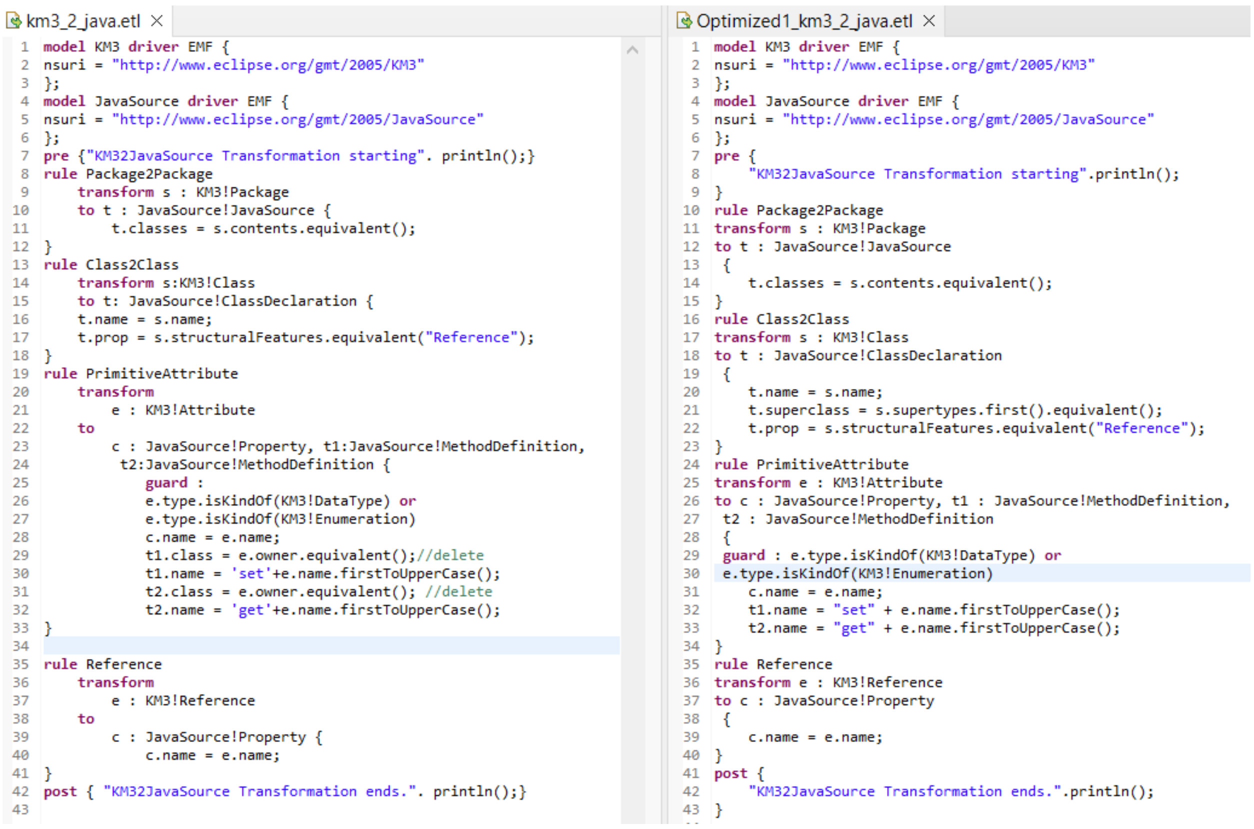
6.3.2 Results

In this subsection, we discuss the obtained results of our experiment, by graphically reporting the results for the first model (original) used as input, and the complete results are reported in Table 6.1. The experiment 1, 2 and 3 reports the results obtained in order to answer research question RQ1, RQ2 and RQ3 respectively.

Experiment 1 An example that shows the difference between the unoptimized and the optimized transformation is shown Fig. 6.5. The target element(s) in the shown transformation is deleted as it would not be required in the next transformation of the chain.

From Fig. 6.5, the resulting models with the optimized and unoptimized transformation resulted in exactly the same model except those modeling elements that are not used in the next transformation. Therefore, the final target model in the optimized transformation chain would generate

[3]https://www.eclipse.org/epsilon/doc/emg/


km3_2_java.etl

```
model KM3 driver EMF {
nsuri = "http://www.eclipse.org/gmt/2005/KM3"
};
model JavaSource driver EMF {
nsuri = "http://www.eclipse.org/gmt/2005/JavaSource"
};
pre {"KM32JavaSource Transformation starting". println();}
rule Package2Package
    transform s : KM3!Package
    to t : JavaSource!JavaSource {
        t.classes = s.contents.equivalent();
}
rule Class2Class
    transform s:KM3!Class
    to t: JavaSource!ClassDeclaration {
    t.name = s.name;
    t.prop = s.structuralFeatures.equivalent("Reference");
}
rule PrimitiveAttribute
    transform
        e : KM3!Attribute
    to
        c : JavaSource!Property, t1:JavaSource!MethodDefinition,
         t2:JavaSource!MethodDefinition {
            guard :
            e.type.isKindOf(KM3!DataType) or
            e.type.isKindOf(KM3!Enumeration)
            c.name = e.name;
            t1.class = e.owner.equivalent();//delete
            t1.name = 'set'+e.name.firstToUpperCase();
            t2.class = e.owner.equivalent(); //delete
            t2.name = 'get'+e.name.firstToUpperCase();
}

rule Reference
    transform
        e : KM3!Reference
    to
        c : JavaSource!Property {
            c.name = e.name;
}
post { "KM32JavaSource Transformation ends.". println();}
```

Optimized1_km3_2_java.etl

```
model KM3 driver EMF {
nsuri = "http://www.eclipse.org/gmt/2005/KM3"
};
model JavaSource driver EMF {
nsuri = "http://www.eclipse.org/gmt/2005/JavaSource"
};
pre {
    "KM32JavaSource Transformation starting".println();
}
rule Package2Package
transform s : KM3!Package
to t : JavaSource!JavaSource
 {
    t.classes = s.contents.equivalent();
}
rule Class2Class
transform s : KM3!Class
to t : JavaSource!ClassDeclaration
 {
    t.name = s.name;
    t.superclass = s.supertypes.first().equivalent();
    t.prop = s.structuralFeatures.equivalent("Reference");
}
rule PrimitiveAttribute
transform e : KM3!Attribute
to c : JavaSource!Property, t1 : JavaSource!MethodDefinition,
 t2 : JavaSource!MethodDefinition
 {
 guard : e.type.isKindOf(KM3!DataType) or
 e.type.isKindOf(KM3!Enumeration)
    c.name = e.name;
    t1.name = "set" + e.name.firstToUpperCase();
    t2.name = "get" + e.name.firstToUpperCase();
}
rule Reference
transform e : KM3!Reference
to c : JavaSource!Property
 {
    c.name = e.name;
}
post {
    "KM32JavaSource Transformation ends.".println();
}
```

FIGURE 6.5: Unoptimized and optimized KM32JavaSource transformation example

an exact target model as in the case of unoptimized chain. Also, the optimization algorithm 2 would move back through the chain starting from the last transformation of the chain till the first transformation of the chain. That's why the target model would be preserved in the optimized chain as well. However, the intermediate transformation(s) of the optimized chain may lead to different model(s) when compared to the unoptimized chain. After using EMFCompare on the target models used from optimized and unoptimized chain, there are no difference between these two target models.

Experiment 2 The experiment for the original model are reported in the `Difference` column of Table 6.1. Based on the user input, two available transformation chains are returned by the algorithm with their respective optimized version of the transformation chain.

CH_1: $KM3 \rightarrow EMF \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

CH_2: $KM3 \rightarrow JavaSource \rightarrow Table \rightarrow HTML \rightarrow XML$

We have reported the execution time for the entire chains $CH1$ and $CH2$ on the last column, and the result confirms the reduction of 17.44% of CH_1 and 1.27% of CH_2. It has been observed that the optimization algorithm would perform better if the no. of deleted modelling elements is more for a relatively longer transformation chain. The results confirm that the approach has been able to optimize the execution time, still producing correct models in output, and this confirms the results for RQ1 and RQ2.

Experiment 3 Table 6.1 and 6.2 reports the complete results of the execution time and the generated target elements of the experiment.

		Execution time					
		Unoptimized		Optimized		Difference	
IN model	size	CH_1	CH_2	CH_1	CH_2	CH_1	CH_2
original_4	788	2.94	2.33	2.89	2.27	0.05	0.0508
generated #1	1150	4.92	2.53	4.24	2.46	0.68	0.0746
generated #2	2300	6.43	2.58	5.84	2.57	0.59	0.002
generated #3	3450	8.51	2.60	8.29	2.59	0.22	0.003
generated #4	4600	14.61	2.73	13.12	2.67	1.49	0.0574
generated #5	5750	30.89	2.90	22.95	2.85	7.94	0.0562
generated #6	6300	37.46	2.86	33.15	2.84	4.31	0.0224
generated #7	7450	49.12	3.12	44.40	3.03	4.72	0.088
generated #8	9200	65.55	3.26	63.40	3.20	2.15	0.058
generated #9	10350	82.45	3.27	79.07	3.25	3.38	0.0226
generated #10	11500	99.86	3.38	83.45	3.28	16.41	0.1002
generated #11	13800	129.14	4.37	120.59	3.83	8.55	0.538
generated #12	16100	183.74	4.14	180.57	3.87	3.17	0.2608
generated #13	18400	1239.73	4.62	182.67	4.34	57.06	0.2804
generated #14	20700	262.39	5.35	246.90	4.66	15.494	0.6846
generated #15	23000	360.87	5.04	332.87	4.89	28.34	0.1466

TABLE 6.1: Results for the execution time KM32XML chain experiment

		Total targets generated					
		Unoptimized		Optimized		Difference	
IN model	size	CH_1	CH_2	CH_1	CH_2	CH_1	CH_2
original_4	788	3690	778	3512	697	178	81
generated #1	1150	35186	11894	25540	11745	9646	149
generated #2	2300	59036	23694	48240	23395	10796	299
generated #3	3450	92992	35494	76796	35045	16196	449
generated #4	4600	123858	47294	102262	46695	21596	599
generated #5	5750	154754	59094	127758	58345	26996	749
generated #6	6300	185818	70894	153422	69995	32396	899
generated #7	7450	211952	82694	178906	81645	33046	1049
generated #8	9200	247490	94494	204294	93295	43196	1199
generated #9	10350	278632	106294	230036	104945	48596	1349
generated #10	11500	309582	118094	255586	116595	53996	1499
generated #11	13800	371224	141694	306428	139895	64796	1799
generated #12	16100	433148	165294	357552	163195	75596	2099
generated #13	18400	494964	188894	408568	186495	86396	2399
generated #14	20700	557158	212494	459962	209795	97196	2699
generated #15	23000	698944	236094	510948	233095	107996	2999

TABLE 6.2: Results for total target elements generated in the KM32XML chain experiment

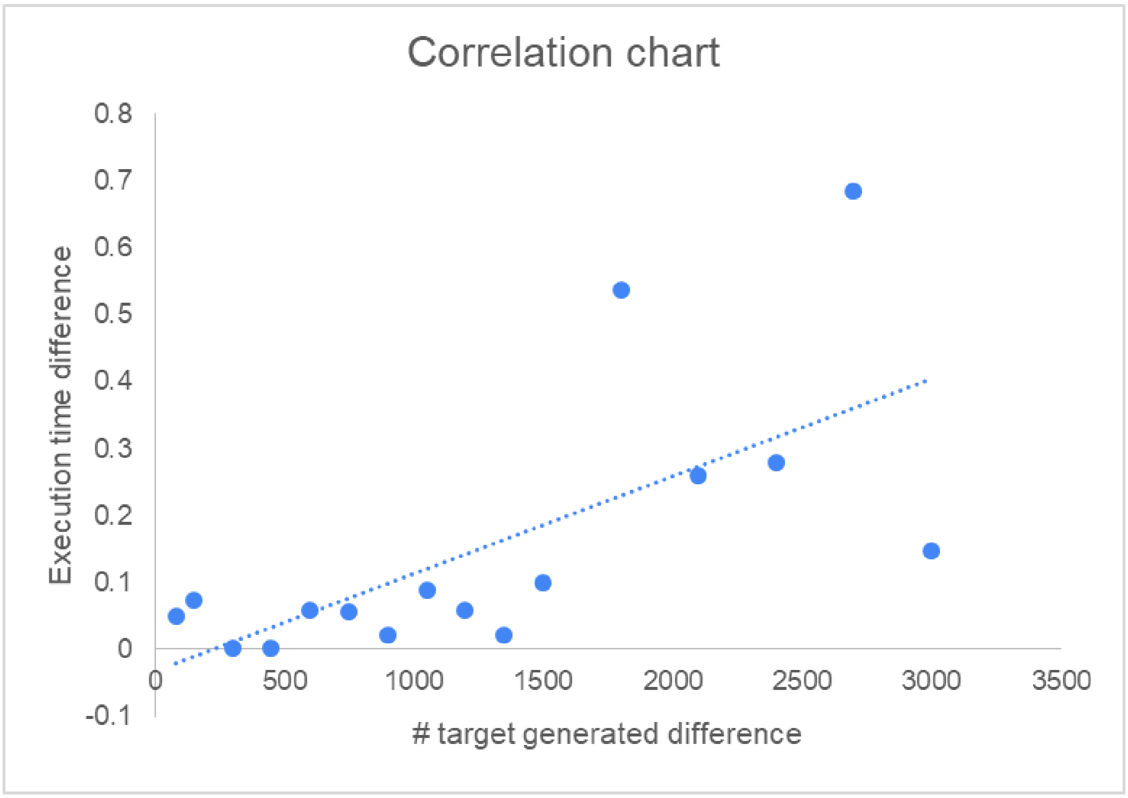


FIGURE 6.6: Correlation between execution time and generated target elements for chain Ch_1

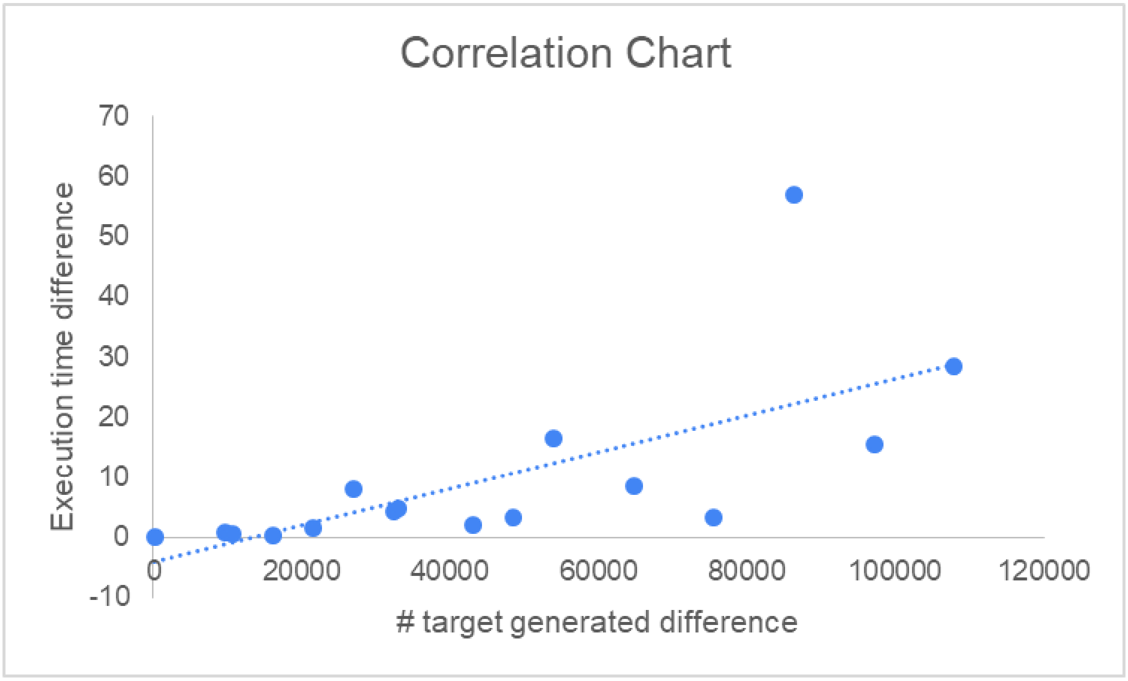


FIGURE 6.7: Correlation between execution time and generated target elements for chain Ch_2

Fig. 6.6 and 6.7 represent the scatter plot for the unoptimized and the optimized chains CH_1 and chain CH_2 experimentation, respectively. The scatter plot shows the correlation between the difference in the generated target elements and the the difference in the execution time for chain CH_1 and chain CH_2 respectively. The chain CH_1 and CH_2 has been executed for 16 different models. The optimized chain CH_1 gives better result (lesser execution time) than the normal chain and larger difference in the generated target elements. While there are much lesser difference in calculating the execution time of the optimized chain CH_2, there are much lower difference in generated target elements. The correlation between the target elements and the execution time difference for CH_1 is 0.678 and that of CH_2 is 0.67. This positive correlation points out the dependence of the generated target elements and it's execution time which confirms the research question RQ3. This signifies that the generated targeted elements in executing a transformation chain is directly proportional to its execution time.

6.4 Threats to validity

In this section we discuss the threats to validity by dividing them as internal and external.

Internal threats Internal threats are aspects influencing the results of the evaluation. One of the aspects that needs to be mentioned is that not all the transformation constructs are considered in the analysis. Overall, the optimization algorithm takes note on each statements within the transformation rule of the transformation. The EMF API such as .eContainer(), .eResource() etc. and greedy/lazy rules and rule inheritance are not considered in the algorithm. The `operation` in ETL files are not considered for optimizing the transformation chain. Also, the elements such as `annotations` and `extends` need to be written manually in the optimized transformation if they exist in the normal transformation. Moreover, executing transformation chains on a single machine can be influenced by other tasks in execution. For this reason, we have executed the chains for both the unoptimized and optimized version 10 times and then used the average of the results. As the optimization is defined as the use of structural elements in the binding of the transformation to a binding of the next transformation, it is henceforth evident that there won't be any optimization case for the last transformation in a chain or even in case of the direct transformation. The dataset of models used for the experiment is composed of 16 models, that could be seen as a limited size for a dataset, but it is quite hard to find online resources in which we have model transformations and available chains for them. For this reason, we tried to mitigate this threat by randomly generating models from a given seed model.

External threats The external factors influencing the conducted experiment's validity outside the used setting are multiple. We tested the approach on the Epsilon framework, and specifically with ETL transformations, but the generalizability of the approach is based on the fact that ETL is a declarative rule-based transformation language and thus all transformation languages falling in this category are candidates for applying this approach. The static analyzer must be re-implemented in order to be able to analyze other types of transformations, e.g., ATL.

6.5 Summary

In our proposal, we present an approach to optimize transformation chains with the aim of accelerating their execution. We argue that by conducting a static analysis of the transformation chain, we can identify the transformation rules that solely generate the necessary intermediate model elements. Through automated program rewriting, we create an optimized program that retains only these identified rules. As a result, the execution time of the original transformation chain is significantly reduced. Furthermore, we also examine the structural features of the entire chain to assess its complexity. This investigation allows us to achieve the desired outcome of optimizing execution while preserving the constraints and properties of the original transformations. By employing this approach, we can enhance the overall efficiency and performance of transformation chains, leading to faster and more streamlined execution.

Chapter 7

Conclusion

7.1 Summary of the contributions

All the work done during my Ph.D. addressing the challenges and objectives in Chapter 1 are highlighted below.

7.1.1 Elaborating the features of various low-code development platforms

Low-code platforms are equipped with a range of features, including intuitive drag-and-drop interfaces, ready-made templates, and powerful visual modeling tools. These user-friendly tools enable individuals to seamlessly create applications, even in the absence of extensive programming expertise [97]. The investigation and findings related to this topic are extensively discussed in Chapter 3.

7.1.2 Analyzing the business process and data handling capabilities of different low-code development platforms

When analyzing the business process and data handling capabilities of different low-code platforms, it is important to consider the organisation's specific needs. For example, some platforms may be better suited for automating complex business processes, while others may focus more on data handling and integration. Additionally, it is important to consider the scalability and security of the platform, as well as the level of support and training offered by the vendor. Therefore, understanding the features of low-code platforms can help organizations to effectively leverage these tools to improve the efficiency and effectiveness of their business processes and data handling capabilities [93]. This work is discussed in Chapter 4.

7.1.3 Applying search-based optimization to search and chain model transformations

The algorithm for identifying model transformation chains is a process that uses a set of rules and steps to identify the sequence of transformations applied to an input model to generate a target one according to user goals. The algorithm starts by analyzing the input and output models and then uses various techniques to determine the transformations that must be applied. Once the algorithm has identified the transformation chain, it can select the best transformation chain and then optimize it to improve its execution time and generated target elements [94].

Search-based optimization techniques are used to select the optimum transformation chain when applying model transformations. These techniques involve using algorithms, such as genetic algorithms or random search, to search for the best possible sequence of transformations that will produce the desired output [95]. The process used in the MOMoT framework [44] begins by defining the objectives of the transformation, such as reducing complexity or improving transformation coverage, etc. Then, a set of potential transformation sequences is generated and evaluated using a fitness function. The fitness function determines how well each sequence of transformations meets the defined objectives.

As the search algorithm runs, it explores different possible sequences and updates the best sequence found, called the best candidate solution. The algorithm continues running until it reaches the stopping criterion, such as reaching a certain number of iterations or finding a candidate solution that meets the desired objectives. For example, the stopping criteria could be reaching the user-defined target model. Search-based optimization techniques are particularly useful when dealing with complex and large transformations, as they can explore many possibilities in a relatively short time. Additionally, these techniques can be easily adapted to consider different constraints and requirements.

Thus, applying search-based optimization techniques to select the optimum transformation chain can help users to improve the efficiency and effectiveness of their model transformation processes. It allows for finding the best possible sequence of transformations that will produce the desired output while considering the user's specific requirements and constraints. This complete work is described in Chapter 5.

7.1.4 Optimization of model transformation chain executions

To improve the efficiency and effectiveness of model transformation composition, the optimization of the execution of the best available transformation chain is proposed. The optimization is done by using only those model elements transformed throughout the chain that are useful to generate the target model. This process would require constant monitoring as some dependent transformation rules may exist. However, the process of optimizing the execution of transformation chains helps to improve the applicability of model transformation composition in application development tools such as low-code development platforms [94]. This optimization work is explained in Chapter 6.

7.2 Publications

The publications which are published during this PhD are as follows.

1. Sahay, Apurvanand, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. "Analyzing business process management capabilities of low-code development platforms." Software: Practice and Experience (2022) [93]

2. Sahay, Apurvanand, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. "Supporting the understanding and comparison of low-code development platforms." In 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 171-178. IEEE, 2020 [97]

3. Sahay, Apurvanand, Davide Di Ruscio, and Alfonso Pierantonio. "Understanding the role of model transformation compositions in low-code development platforms." In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1-5. 2020 [92]

Apart from the above publications, the following two research articles are under submission.

1. Sahay, Apurvanand, Martin Eisenberg, Davide Di Ruscio, Ludovico Iovino, Manuel Wimmer and Alfonso Pierantonio. “Selecting the optimum model transformation chains with MOMoT”. In: to be submitted in a Journal (2023), pp. 1–24 [95]

2. Sahay, Apurvanand, Qurat ul ain Ali, Davide Di Ruscio, Ludovico Iovino, Dimitris Kolovos, Konstantinos Barmpis and Alfonso Pierantonio. "Optimizing the execution of the model transformation chain". In: to be submitted in a Journal (2023), pp. 1–20 [94]

7.3 Developed tools

The developed tools are shown in the GitHub repositories below.

1. The code for selecting model transformation chains with MOMoT is shown in the GitHub link `https://github.com/lowcomote/chainselection_momot.git` and explained in Chapter 5.

2. The code for optimizing the execution of the transformation chain is shown in the GitHub link `https://github.com/lowcomote/chain-optimisation.git` and explained in Chapter 6.

7.4 Future work

In the future, we plan to employ the proposed conceptual framework to analyze additional low-code platforms. We expect a consequent refinement of the conceptual framework shown in Chapter 3. Moreover, we plan to focus more on the reusability and interoperability facilities that low-code development platforms need. The main goals are *i)* the design and development of a repository supporting the reuse of already developed low-code artifacts, *ii)* and development of generic mechanisms enabling the interoperability of different platforms.

The work of optimizing the execution of model transformation chains shown in Chapter 6 can be extended in a future iteration to enhance the proposed approach to exploit other common optimization patterns. Also, the proposed approach can be tested with more complex transformation chains. By benchmarking this approach on transformation chains for large-scale models, some performance results can be produced to demonstrate performance improvements.

Appendix A

Used technologies

The major technologies used throughout the PhD thesis are listed below.

- *EMF Modeling Technologies* - In Chapter 5, the metamodels and models are built using EMF technologies.
  - Using tree-like Ecore editor - It helps to build a tree-like structure of a metamodel defining a domain-specific area.
  - Using GenModel to generate code from ecore file - Genmodel used to execute the metamodel file into its corresponding Java code.
- *Epsilon Languages* - This domain-specific language family is used in Chapter 5 and Chapter 6.
  - *Epsilon Object Language (EOL)* - We used EOL and its static analyser to identify different modelling elements used in a model. This helps to find out information about the model such as model coverage.
  - *Epsilon Transformation Langauge (ETL)* - We use ETL and its static analyser to build and identify various modelling elements and transformation rules defined in a model transformation. ETL language and its static analyser is used to build an ETL model transformation while defining the transformation coverage and analysing the rules and bindings used to optimize the transformation.
  - *Epsilon Model Generation (EMG)* - EMG is used to create multiple models of different modeling instances with different sizes. It is useful for testing the optimization approach for smaller and larger models.
- *MOMoT Technologies* - As described in Chapter 5, MOMoT is a framework that we used to transform an optimization problem into a model-driven problem and then run various optimization algorithms for search-based (model-driven) problems.
  - *XText programming language* - XText is used to write various objectives, constraints, algorithms and other search-based configurations to program a search-based optimization over the chain selection problem.
  - *Henshin model transformation language* - Henshin is used to graphically design a chain selection problem where an in-place transformation is used to model various transformation chains between a source and target metamodel.
- *Java programming language* - We use Java to code all the chain identification algorithms, fitness functions in chain selection and chain optimization process defined in Chapter 5 and Chapter 6.

Bibliography

[1] Lorenzo Addazi et al. "Semantic-based Model Matching with EMFCompare." In: *Me@ models*. 2016, pp. 40–49.

[2] Santiago Aguirre and Alejandro Rodriguez. "Automation of a business process using robotic process automation (RPA): A case study". In: *Workshop on engineering applications*. Springer. 2017, pp. 65–71.

[3] Qurat ul ain Ali and Apurvanand Sahay. *Chain optimisation code*. Version 1.0.1. Dec. 2022. URL: https://github.com/lowcomote/chain-optimisation.git.

[4] Qurat ul ain Ali, Dimitris Kolovos, and Konstantinos Barmpis. "Efficiently Querying Large-Scale Heterogeneous Models". In: *Proc. of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. MODELS '20. Virtual Event, Canada: Association for Computing Machinery, 2020. ISBN: 9781450381352. DOI: 10.1145/3417990.3420207.

[5] Qurat ul ain Ali, Dimitris Kolovos, and Konstantinos Barmpis. "Efficiently querying large-scale heterogeneous models". In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2020, pp. 1–5.

[6] Thomas Allweyer. *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2016.

[7] Camilo Alvarez and Rubby Casallas. "MTC Flow: A tool to design, develop and deploy model transformation chains". In: *Proceedings of the workshop on ACadeMics Tooling with Eclipse*. 2013, pp. 1–9.

[8] *Amazon Honeycode Capabilities*. https://www.honeycode.aws/features. Accessed: 2021-03-01.

[9] Marcel F van Amstel and Mark GJ Van Den Brand. "Model transformation analysis: Staying ahead of the maintenance nightmare". In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2011, pp. 108–122.

[10] *An Introduction to Low-Code Platform*. https://www.mendix.com/low-code-guide/. Accessed: 2020-03-23.

[11] Tony Andrews et al. *Business process execution language for web services*. 2003.

[12] *Appian Platform Overview*. https://www.appian.com/. Accessed: 2020-03-23.

[13] Vincent Aranega, Anne Etien, and Sebastien Mosser. "Using feature model to build model transformation chains". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2012, pp. 562–578.

[14] Thorsten Arendt et al. "Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations". In: *Model Driven Engineering Languages and Systems - 13th International Conference, MODELS 2010, Oslo, Norway, October 3-8, 2010, Proceedings, Part I*. Ed. by Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen. Vol. 6394. Lecture Notes in Computer Science. Springer, 2010, pp. 121–135. DOI: 10.1007/978-3-642-16145-2_9. URL: https://doi.org/10.1007/978-3-642-16145-2%5C_9.

[15] Thorsten Arendt et al. “Henshin: advanced concepts and tools for in-place EMF model transformations”. In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2010, pp. 121–135.

[16] *ATL Transformations*. https://www.eclipse.org/atl/atlTransformations/. Accessed: 2022-05-28.

[17] Francesco Basciani, Ludovico Iovino, Alfonso Pierantonio, et al. “MDEForge: an extensible web-based modeling platform”. In: *2nd International Workshop on Model-Driven Engineering on and for the Cloud, CloudMDE 2014, Co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2014*. Vol. 1242. CEUR-WS. 2014, pp. 66–75.

[18] Francesco Basciani et al. “A tool for automatically selecting optimal model transformation chains”. In: *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2018, pp. 2–6.

[19] Francesco Basciani et al. “Automated selection of optimal model transformation chains via shortest-path algorithms”. In: *IEEE Transactions on Software Engineering* 46.3 (2018), pp. 251–279.

[20] Francesco Basciani et al. “Exploring model repositories by means of megamodel-aware search operators.” In: *MoDELS (Workshops)*. 2018, pp. 793–798.

[21] Francesco Basciani et al. “MDEForge: An extensible Web-based modeling platform”. In: vol. 1242. Sept. 2014.

[22] Francesco Basciani et al. “Model repositories: Will they become reality?” In: *CloudMDE@ MoDELS*. 2015, pp. 37–42.

[23] Eduard Bauer, Jochen M Küster, and Gregor Engels. “Test suite quality for model transformation chains”. In: *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer. 2011, pp. 3–19.

[24] Reda Bendraou et al. “A comparison of six uml-based languages for software process modeling”. In: *IEEE Transactions on Software Engineering* 36.5 (2010), pp. 662–675.

[25] *Best Low-Code Development Platforms Software*. https://www.g2.com/categories/low-code-development-platforms. Accessed: 2020-05-26.

[26] Jean Bézivin. “Model driven engineering: An emerging technical space”. In: *Generative and Transformational Techniques in Software Engineering: International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers* (2006), pp. 36–64.

[27] Robert Bill et al. “A local and global tour on MOMoT”. In: *Softw. Syst. Model.* 18.2 (2019), pp. 1017–1046. DOI: 10.1007/s10270-017-0644-3. URL: https://doi.org/10.1007/s10270-017-0644-3.

[28] Dominik Birkmeier and Sven Overhage. “Is BPMN really first choice in joint architecture development? an empirical study on the usability of BPMN and UML activity diagrams for business users”. In: *International conference on the quality of software architectures*. Springer. 2010, pp. 119–134.

[29] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Vol. 1. Sept. 2012. DOI: 10.2200/S00441ED1V01Y201208SWE001.

[30] Jordi Cabot and Martin Gogolla. "Object Constraint Language (OCL): A Definitive Guide". In: *Formal Methods for Model-Driven Engineering: 12th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2012, Bertinoro, Italy, June 18-23, 2012. Advanced Lectures*. Ed. by Marco Bernardo, Vittorio Cortellessa, and Alfonso Pierantonio. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 58–90. ISBN: 978-3-642-30982-3. DOI: 10.1007/978-3-642-30982-3_3. URL: https://doi.org/10.1007/978-3-642-30982-3_3.

[31] Jordi Cabot and Ernest Teniente. "A metric for measuring the complexity of OCL expressions". In: *Model Size Metrics Workshop co-located with MODELS*. Vol. 6. Citeseer. 2006, p. 10.

[32] Flavio Corradini et al. "A guidelines framework for understandable BPMN models". In: *Data & Knowledge Engineering* 113 (2018), pp. 129–154.

[33] Krzysztof Czarnecki. "Domain Engineering". In: *Encyclopedia of Software Engineering*. American Cancer Society, 2002, pp. 433–444. ISBN: 9780471028956. DOI: 10.1002/0471028959.sof095. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/0471028959.sof095. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/0471028959.sof095.

[34] Krzysztof Czarnecki. "Generative programming - principles and techniques of software engineering based on automated configuration and fragment-based component models". PhD thesis. Technische Universität Illmenau, Germany, 1999. URL: http://d-nb.info/958706700.

[35] Kalyanmoy Deb and Himanshu Jain. "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints". In: *IEEE transactions on evolutionary computation* 18.4 (2013), pp. 577–601.

[36] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197.

[37] Juri Di Rocco et al. "Systematic recovery of MDE technology usage". In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2018, pp. 110–126.

[38] Juri Di Rocco et al. "Understanding MDE projects: megamodels to the rescue for architecture recovery". In: *Software and Systems Modeling* 19.2 (2020), pp. 401–423.

[39] JM Dorador and Robert IM Young. "Application of IDEF0, IDEF3 and UML methodologies in the creation of information models". In: *International Journal of Computer Integrated Manufacturing* 13.5 (2000), pp. 430–445.

[40] Sven Efftinge et al. "Xbase: implementing domain-specific languages for Java". In: *Generative Programming and Component Engineering, GPCE'12, Dresden, Germany, September 26-28, 2012*. Ed. by Klaus Ostermann and Walter Binder. ACM, 2012, pp. 112–121. DOI: 10.1145/2371401.2371419. URL: https://doi.org/10.1145/2371401.2371419.

[41] Martin Eisenberg et al. "Towards Reinforcement Learning for In-Place Model Transformations". In: *24th International Conference on Model Driven Engineering Languages and Systems, MODELS 2021, Fukuoka, Japan, October 10-15, 2021*. IEEE, 2021, pp. 82–88. DOI: 10.1109/MODELS50736.2021.00017. URL: https://doi.org/10.1109/MODELS50736.2021.00017.

[42] Hüseyin Ergin and Eugene Syriani. "Identification and application of a model transformation design pattern". In: *ACM southeast conference, ACMSE*. Vol. 13. 2013.

[43] Anne Etien et al. "Chaining model transformations". In: *Proceedings of the First Workshop on the Analysis of Model Transformations*. 2012, pp. 9–14.

[44] Martin Fleck, Javier Troya, and Manuel Wimmer. "Marrying search-based optimization and model transformation technology". In: *Proc. of NasBASE* (2015), pp. 1–16.

[45] Martin Fleck, Javier Troya, and Manuel Wimmer. "Search-based model transformations with MOMoT". In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2016, pp. 79–87.

[46] Martin Fleck et al. "Model transformation modularization as a many-objective optimization problem". In: *IEEE Transactions on Software Engineering* 43.11 (2017), pp. 1009–1032.

[47] UML Revision Task Force. "OMG unified modeling language: Superstructure". In: *Object Management Group (OMG)* (2010).

[48] Robert France, Jim Bieman, and Betty HC Cheng. "Repository for model driven development (ReMoDD)". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2006, pp. 311–317.

[49] Domıénguez-Mayo Garcıéa-Garcıéa Julián Alberto; Enrıéquez; José Gonzalez. "Characterizing and evaluating the quality of software process modeling language: Comparison of ten representative model-based languages". In: *Computer Standards & Interfaces* 63 (2019), pp. 52–66.

[50] *Gartner Forecasts*. https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-2021. Accessed: 2022-10-28.

[51] Diimitrios Georgakopoulos, Mark Hornick, and Amit Sheth. "An overview of workflow management: From process modeling to workflow automation infrastructure". In: *Distributed and parallel Databases* 3.2 (1995), pp. 119–153.

[52] *Google App Maker Platform guide*. https://developers.google.com/appmaker/overview. Accessed: 2020-03-23.

[53] *Google Appsheet Actions*. https://help.appsheet.com/en/articles/953637-actions-the-essentials. Accessed: 2022-11-09.

[54] Khouloud Guizani and Sonia Ayachi Ghannouchi. "An approach for selecting a business process modeling language that best meets the requirements of a modeler". In: *Procedia Computer Science* 181 (2021), pp. 843–851.

[55] Mariam Ben Hassen, Mohamed Turki, and FaıHERE!HERE!ez Gargouri. "Choosing a sensitive business process modeling formalism for knowledge identification". In: *Procedia Computer Science* 100 (2016), pp. 1002–1015.

[56] Sorour Jahanbin. "Efficient Model Loading through Static Analysis". In: *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2021, pp. 660–665.

[57] Frédéric Jouault et al. "ATL: A model transformation tool". In: *Science of computer programming* 72.1-2 (2008), pp. 31–39.

[58] Jonathan Keel. *Salesforce.com Lightning Process Builder and Visual Workflow*. Springer, 2016.

[59] Zador Daniel Kelemen et al. *Selecting a process modeling language for process based unification of multiple standards and models*. Tech. rep. Budapest, Technical Report TR201304, 2013.

[60] Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. "Model transformation as an optimization problem". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2008, pp. 159–173.

[61] Marouane Kessentini et al. "Search-based model transformation by example". In: *Software & Systems Modeling* 11.2 (2012), pp. 209–226.

[62] *Kissflow Platform Overview*. https://kissflow.com/process-management/. Accessed: 2020-03-23.

[63] Dimitrios S Kolovos, Richard F Paige, and Fiona AC Polack. “The epsilon transformation language”. In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2008, pp. 46–60.

[64] Dimitris Kolovos et al. *The Epsilon Book. Eclipse*. 2010.

[65] Jochen M Küster, Thomas Gschwind, and Olaf Zimmermann. “Incremental development of model transformation chains using automated testing”. In: *International conference on model driven engineering languages and systems*. Springer. 2009, pp. 733–747.

[66] Maciej Laszczyk and Paweł B Myszkowski. “Improved selection in evolutionary multi-objective optimization of multi-skill resource-constrained project scheduling problem”. In: *Information Sciences* 481 (2019), pp. 412–431.

[67] Théo Le Calvar et al. “Efficient ATL Incremental Transformations”. In: *The Journal of Object Technology* 18 (July 2019), 2:1. DOI: 10.5381/jot.2019.18.3.a2.

[68] Levi Lúcio et al. “FTG+ PM: An integrated framework for investigating model transformation chains”. In: *International SDL Forum*. Springer. 2013, pp. 182–202.

[69] Stephen MacDonell et al. “How reliable are systematic reviews in empirical software engineering?” In: *IEEE Transactions on Software Engineering* 36.5 (2010), pp. 676–687.

[70] Ricardo Martins et al. “An overview on how to develop a low-code application using OutSystems”. In: *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*. IEEE. 2020, pp. 395–401.

[71] Jacqueline A McQuillan and James F Power. “White-box coverage criteria for model transformations”. In: *Proceedings of the 1st International Workshop on Model Transformation with ATL*. Springer-Verlag LNCS series. 2009, pp. 63–77.

[72] *Mendix Microflow*. https://docs.mendix.com/refguide/microflows. Accessed: 2022-11-09.

[73] *Mendix Platform Features*. https://www.mendix.com/platform/. Accessed: 2020-03-23.

[74] *Microsoft PowerApps flow*. https://docs.microsoft.com/en-us/powerapps/user/use-flows. Accessed: 2022-11-09.

[75] *Microsoft PowerApps Platform Overview*. https://docs.microsoft.com/en-us/powerapps/maker/. Accessed: 2020-03-23.

[76] Mohamed Wiem Mkaouer and Marouane Kessentini. “Model transformation using multi-objective optimization”. In: *Advances in Computers*. Vol. 92. Elsevier, 2014, pp. 161–202.

[77] Gunter Mussbacher et al. “Opportunities in intelligent modeling assistance”. In: *Software and Systems Modeling* 19.5 (2020), pp. 1045–1053.

[78] Justice Opara-Martins, R. Sahandi, and Feng Tian. “Implications of Integration and Interoperability for Enterprise Cloud-Based Applications”. In: Oct. 2015, pp. 213–223. DOI: 10.1007/978-3-319-38904-2_22.

[79] *OutSystem Platform Business Process*. https://success.outsystems.com/Documentation/11/Developing_an_Application/Use_Processes_(BPT). Accessed: 2022-11-09.

[80] *OutSystem Platform Features*. https://www.outsystems.com/platform/. Accessed: 2020-03-23.

[81] Chun Ouyang et al. “From business process models to process-oriented software systems”. In: *ACM transactions on software engineering and methodology (TOSEM)* 19.1 (2009), pp. 1–37.

[82] José Luıés Pereira and Diogo Silva. “Business process modeling languages: A comparative framework”. In: *New Advances in Information Systems and Technologies*. Springer, 2016, pp. 619–628.

[83] Carl Adam Petri and Wolfgang Reisig. “Petri net”. In: *Scholarpedia* 3.4 (2008), p. 6477.

[84] Saheed Popoola, Dimitrios S Kolovos, and Horacio Hoyos Rodriguez. “EMG: A domain-specific transformation language for synthetic model generation”. In: *International Conference on Theory and Practice of Model Transformations*. Springer. 2016, pp. 36–51.

[85] Carlos Portela et al. “A comparative analysis between BPMN and SPEM modeling standards in the software processes context”. In: *Journal of Software Engineering and Applications 05(05):330-339* (2012).

[86] Ali Razavi and Kostas Kontogiannis. “Partial evaluation of model transformations”. In: *2012 34th International Conference on Software Engineering (ICSE)*. 2012, pp. 562–572. DOI: 10.1109/ICSE.2012.6227160.

[87] Jan Recker et al. “Business Process Modeling- A Comparative Analysis”. In: *Journal of the Association of Information Systems* 10 (Apr. 2009). DOI: 10.17705/1jais.00193.

[88] Wolfgang Reisig. *Petri nets: an introduction*. Vol. 4. Springer Science & Business Media, 2012.

[89] C. Richardson and J. R. Rymer. “The Forrester Wave: Low-Code Development Platforms, Q2 2016. Tech. rep”. In: *Forrester Research* (2016).

[90] Juri Rocco et al. “Collaborative Repositories in Model-Driven Engineering [Software Technology]”. In: *IEEE Software* 32 (May 2015), pp. 28–34. DOI: 10.1109/MS.2015.61.

[91] John R. Rymer. “The Forrester Wave: Low-Code Platforms For Business Developers, Q2 2019”. In: *Forrester Research* (2019).

[92] Apurvanand Sahay, Davide Di Ruscio, and Alfonso Pierantonio. “Understanding the role of model transformation compositions in low-code development platforms”. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2020, pp. 1–5.

[93] Apurvanand Sahay et al. “Analyzing business process management capabilities of low-code development platforms”. In: *Software: Practice and Experience* (2022).

[94] Apurvanand Sahay et al. “Optimizing the execution of the model transformation chain”. In: *to be submitted in journal* (2023), pp. 1–20.

[95] Apurvanand Sahay et al. “Selecting the optimum model transformation chains with MOMoT”. In: *to be submitted in a Journal* (2023), pp. 1–24.

[96] Apurvanand Sahay et al. “Supporting the understanding and comparison of low-code development platforms”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications*. 2020.

[97] Apurvanand Sahay et al. “Supporting the understanding and comparison of low-code development platforms”. In: *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE. 2020, pp. 171–178.

[98] Dilan Sahin et al. “Model transformation testing: a bi-level search-based software engineering approach”. In: *Journal of Software: Evolution and Process* 27.11 (2015), pp. 821–837.

[99] *Salesforce App Cloud Platform Overview*. https://developer.salesforce.com/platform. Accessed: 2020-03-23.

[100] *Salesforce Lightning flow*. https://www.salesforce.com/in/products/platform/solutions/automate-business-processes/. Accessed: 2022-11-09.

[101] Jesús Sánchez Cuadrado et al. “Efficient execution of ATL model transformations using static analysis and parallelism”. In: *IEEE Transactions on Software Engineering* PP (July 2020), pp. 1–1. DOI: 10.1109/TSE.2020.3011388.

[102] Gehan MK Selim, James R Cordy, and Juergen Dingel. “Model transformation testing: The state of the art”. In: *Proceedings of the first workshop on the analysis of model transformations*. 2012, pp. 21–26.

[103] Shane Sendall and Wojtek Kozaczynski. “Model Transformation: The Heart and Soul of Model-Driven Software Development”. In: *Software, IEEE* 20 (Oct. 2003), pp. 42–45. DOI: 10.1109/MS.2003.1231150.

[104] Amandeep Singh, Pardeep Mittal, and Neetu Jha. “FOSS: A Challenge to Proprietary Software”. In: *IJCST* 4 (10 2013).

[105] Hui Song et al. “Applying MDE tools at runtime: Experiments upon runtime models”. In: *Proceedings of the 5th International Workshop on Models at Run Time*. 2010.

[106] *Static Analysis built-on-the-top of Epsilon*. https://github.com/epsilonlabs/static-analysis.git.

[107] Eugene Syriani and Jeff Gray. “Challenges for addressing quality factors in model transformation”. In: *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation*. IEEE. 2012, pp. 929–937.

[108] *Thinkwise Process Flow*. https://docs.thinkwisesoftware.com/docs/2020.1/sf/process_flows.html. Accessed: 2022-11-09.

[109] Massimo Tisi et al. “Lowcomote: Training the next generation of experts in scalable low-code engineering platforms”. In: *STAF 2019 Co-Located Events Joint Proceedings: 1st Junior Researcher Community Event, 2nd International Workshop on Model-Driven Engineering for Design-Runtime Interaction in Complex Systems, and 1st Research Project Showcase Workshop co-located with Software Technologies: Applications and Foundations (STAF 2019)*. 2019.

[110] Javier Troya, Sergio Segura, and Antonio Ruiz-Cortés. “Automated inference of likely metamorphic relations for model transformations”. In: *Journal of Systems and Software* 136 (2018), pp. 188–208.

[111] Qurat Ul Ain Ali, Dimitris Kolovos, and Konstantinos Barmpis. “Selective Traceability for Rule-Based M2M Transformations”. In: *Proceedings of the 15th ACM SIGPLAN International Conference on Software Language Engineering (SLE ’22), December 06-07, 2022, Auckland, New Zealand*. 2022, pp. 751–760. DOI: 10.1145/3567512.3567521.

[112] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. “Hypervolume-based multi-objective reinforcement learning”. In: *International Conference on Evolutionary Multi-Criterion Optimization*. Springer. 2013, pp. 352–366.

[113] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. “Scalarized multi-objective reinforcement learning: Novel design techniques”. In: *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2013, pp. 191–199.

[114] Kristof Van Moffaert and Ann Nowé. “Multi-objective reinforcement learning using sets of pareto dominating policies”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3483–3512.

[115] Daniel Varro et al. “Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework”. In: *Software & Systems Modeling* 15 (July 2016). DOI: 10.1007/s10270-016-0530-4.

[116] Paul Vincent et al. “Magic Quadrant for Enterprise Low-Code Application Platforms”. In: *Gartner report* (2019).

[117] Paul Vincent et al. “Magic quadrant for enterprise low-code application platforms”. In: *Gartner report* (2020).

[118] Dennis Wagelaar. “Composition Techniques for Rule-Based Model Transformation Languages”. In: *Proceedings of the 1st International Conference on Theory and Practice of Model Transformations*. ICMT ’08. Zurich, Switzerland: Springer-Verlag, 2008, pp. 152–167. ISBN: 9783540699262. DOI: 10.1007/978-3-540-69927-9_11. URL: https://doi.org/10.1007/978-3-540-69927-9_11.

[119] Wei Wang et al. “A comparison of business process modeling methods”. In: *2006 IEEE International Conference on Service Operations and Logistics, and Informatics*. IEEE. 2006, pp. 1136–1141.

[120] Robert Waszkowski. “Low-code platform for automating business processes in manufacturing”. In: *IFAC-PapersOnLine* 52.10 (2019), pp. 376–381.

[121] Stephen A White. “Introduction to BPMN”. In: *Ibm Cooperation* 2.0 (2004), p. 0.

[122] Alexander Wise et al. “Using Little-JIL to coordinate agents in software engineering”. In: *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering*. IEEE. 2000, pp. 155–163.

[123] Removed DataTypes from WorkflowProcess, Removed PlainType Element from Schema, and Added Script Element. “Workflow Process Definition Interface—XML Process Definition Language”. In: *Lighthouse Point (Fl): Workflow Management Coalition, (WFMCTC-1025)* (2005).

[124] Zelda B Zabinsky et al. “Random search algorithms”. In: *Department of Industrial and Systems Engineering, University of Washington, USA* (2009).

[125] Dongdai Zhou et al. “The Rete algorithm improvement and implementation”. In: *2008 International Conference on Information Management, Innovation Management and Industrial Engineering*. Vol. 1. IEEE. 2008, pp. 426–429.

[126] *Zoho Creator Deluge Script*. https://www.zoho.com/creator/help/images/intro-to-deluge-workflows.pdf. Accessed: 2022-11-09.

[127] *Zoho Creator Platform Features*. https://www.zoho.com/creator/features.html. Accessed: 2020-03-23.