Received September 21, 2019, accepted October 20, 2019, date of publication November 4, 2019, date of current version November 13, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2950884

A Model Driven Reverse Engineering Framework for Generating High Level UML Models From Java Source Code

UMAIR SABIR, FAROOQUE AZAM, SAMI UL HAQ, MUHAMMAD WASEEM ANWAR, WASI HAIDER BUTT, AND ANAM AMJAD
Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan

Corresponding author: Muhammad Waseem Anwar (waseemanwar@ceme.nust.edu.pk)

ABSTRACT Legacy systems are large applications which are significant in performing daily organizational operations and cannot be upgraded easily especially in the absence of architectural and design documentation. Software modernization is an emerging field of software engineering, which transforms the legacy systems into new one according to the specified requirements of stakeholders. It mainly deals with improving the architecture, features, rules and data sources of existing system. It always remained a challenging task to achieve high-level representation of legacy systems. In order to achieve this high-level representation, Reverse Engineering (RE) plays an integral role. The issues of traditional RE are overcome with the help of Model Driven Reverse Engineering (MDRE) such as it generates model-based view of the legacy systems, which is comprehensible and easy to understand for practitioners. MDRE is an active research area but it provides limited tool support to extract and model both structural and behavioral aspects of legacy systems. In this paper, a novel MDRE framework named as ‘‘**Sour**c**e** to **Mo**del **F**ramework (Src2MoF)’’is proposed to generate Unified Modeling Language (UML) structural (class) and behavioral (activity) diagrams from the Java source code. Src2MoF is based on the principles of Model Driven Engineering (MDE), which use models as first-class citizens alleviating the complexity of software systems. The contributions of this paper are as follows; first, an Intermediate Model Discoverer (IMD) is developed using source code parser to get the intermediate representation of the system from the existing Java code. Second, an open source transformation engine named ‘‘UML model generator’’ is implemented using Java, which takes these intermediate models as input, and produce high-level UML models of the subject legacy system. Finally, the two benchmark case studies are presented to depict the relevance and usability of Src2MoF.

INDEX TERMS Model driven reverse engineering, reverse engineering, UML models, legacy systems, java code.

I. INTRODUCTION

Legacy systems are used by many organizations because they execute critical operations and are non-negligible. Moreover, the high cost of replacing these systems limits the productivity of these organizations. Software modernization transforms legacy systems into modern systems based on the stakeholder’s requirements of an organization. Reverse Engineering (RE) plays an important role in software modernization and its history as old as the computer itself. Initially, it targeted computer hardware modernization but later, with the increasing use of software in 80s, it was dedicated for software systems. Reverse Engineering (RE) is a methodology extensively used to extract the valuable information of the underlying system at higher layer of abstraction using existing legacy system artefacts such as requirements, design, code [1]. By applying RE, it helps the practitioners to deal with legacy systems in following manner: 1) It helps to understand the key issues of an unsuccessful design and subsequently improve the design. 2) It is used to perform migration of systems from old platforms to new i.e. From ’as is’ state into a ’to be’ state. Reverse engineering of system can provide the most current documentation necessary for understanding the existing state of the system. RE helps an

The associate editor coordinating the review of this manuscript and approving it for publication was Xu Chen.

organization when a system is required to interface/work with other systems or products.

Reverse Engineering (RE) is a time consuming and an error-prone process. Model Driven Engineering (MDE) [2] is a promising approach based on the principle "Everything is a model". The combination of reverse engineering and model driven engineering is known as Model Driven Reverse Engineering (MDRE). It has gained significant popularity in the research community during past few years and frequently practiced [3] to solve RE problems. Generally, MDRE is a two-step process [4]; 1) Getting an abstract view (i.e., a model) of the analyzed legacy system from source artefacts. 2) Making use of this model to achieve a specific goal, e.g., re-documenting or re-engineering.

Several issues of traditional RE approaches are resolved using MDRE as follows; MDRE comprises of models to alleviate the complexity of traditional code-based software systems. Thus, structural complexity of these legacy systems is reduced. Models in MDE symbolize information and usage of meta-models [2] in order to define the target models. Technical heterogeneity of legacy system is reduced by introducing homogenous models in MDRE [4]. MDRE solutions are scalable, adaptable, portable and reusable. MDRE directly benefits from the genericity, extensibility, integration, coverage, and automation expertise of MDE technologies to provide a good assistance for reverse engineering. Additionally, Text-to-Model (T2M) transformations [7] in MDRE is available with the help of tools. The resultant models obtained using MDRE can be used for all reverse engineering activities.

In Model Based System Engineering (MBSE), Unified Modeling Language (UML) plays a central part of the design phase. It is highly supportive to verify the design of given system in initial development process [5]. This expedites the system development by reducing the efforts in different Software Development Life Cycle (SDLC) phases such as implementation, testing and maintenance. With the increasing use of UML models in software modeling and model driven engineering, there is a need to obtain these models containing as much information as possible in order to re-new legacy system. By considering the importance of UML models, one of the challenges in MDRE is to provide a framework for complete structural as well as behavioral UML model extraction of legacy system. Several research studies already exist such as tools like MoDisco [6] and transformation languages like Gra2MoL [7] are developed to carry out MDRE but the focus of these researches is only on getting the structure of legacy systems from available source code. Gra2MoL is aimed to extract Knowledge Discovery Metamodel (KDM) from general-purpose languages but no practical evaluation of this language is analyzed. However, this research study guided the researcher to define Text-to-Model (T2M) transformations. Similarly, MoDisco [6] does not provide complete support to MDRE. This tool supports Java, JSP and XML meta-models so generated artefacts are not UML models. Authors intended to discover Java models from Java source code. The most prominent approaches so far, discussed the extraction and assessment of structural aspects from the source artefacts but there is a limited support of tooling [8]. Nevertheless, researchers aimed to cover the gap of extraction and assessment of behavioral aspects from the source artefact, yet there is no tool or technique available for MDRE to facilitate both structural and behavioral aspects of legacy system using UML models.

The objective of this research work is to provide a full MDRE framework (Src2MoF) which overcome the above-mentioned research gaps i.e. Capable of generating behavioral models along with structural models from Java source code. The major contributions of Src2MoF are:

1. **Extraction of Structural aspects from source artefact:** Exploration of subject system and pulling out the features related to static structure, elements and relationships between multiple elements of existing system's design.
2. **Extraction of Behavioral aspects from source artefact:** This contribution aims at exploration of functionality of the source code. It fetches the aspects regarding operations, implementation of operations, functions and the function behaviors from source artefacts of existing system.
3. **Discovery of Intermediate Model:** To pile the information extracted about structural and behavioral aspects of source artefacts and make it utilizable, our anticipated approach will be able to format this information using Intermediate Model (IM), which is the depiction of source artefact in XML modeling format.
4. **Transformation:** To illustrate the intermediate model at some higher abstraction level, transformation engine is developed names as 'UML Model Generator'. It is based on Text-to-Model (T2M) transformation implemented in Eclipse using Java language.

To fulfill above mentioned research objectives, we have implemented a complete model driven reverse engineering framework" Src2MoF (**So**urce **to** **Mo**del **F**ramework)" as shown in **FIGURE 1**. Proposed framework encompasses two phases i.e. Discovery of Intermediate Model(s) and Transformation to UML model(s). This leads to assemble a framework comprises two components: 1) Intermediate Model Discoverer (IMD) and 2) UML Model Generator. IMD is responsible for creation of textual model (i.e. XML) by analyzing the source code. Source artefact i.e. Java code, is examined by IMD to figure out a graphical view of textual code. Inside IDM, a source code parser is implemented. This parser interprets the source code in an easy to understand format. We integrated this parser in such way that it creates an *Abstract Syntax Tree (AST)* instead of parsed tree. In Model Discoverer phase, information about structure and behavior of legacy system is extracted from AST, generated in previous step. An intermediate model IM is generated by using this extracted information.

UML Model Generator is a type of transformation engine implemented in *Model Development Tools* which interprets the Intermediate Model (IM) into a standardized format

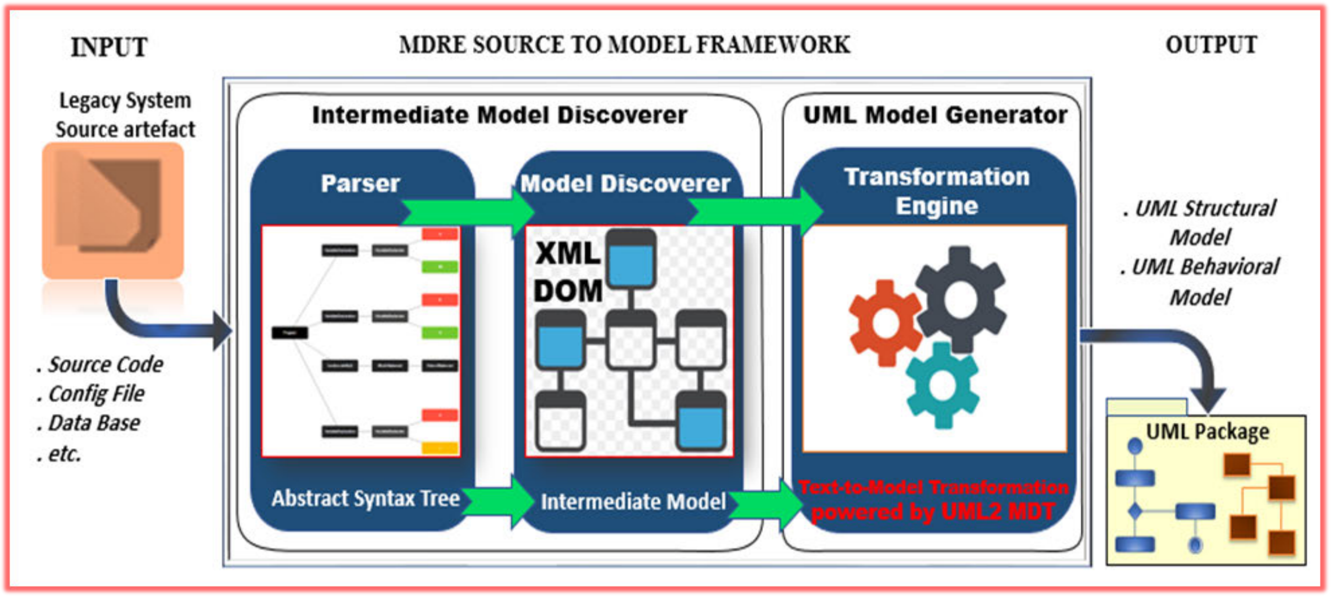


FIGURE 1. **Overview of proposed framework.**

i.e. UML. This engine takes templates (*Transformation Rules)* to transform IMs into UML structural and behavioral models. Src2MoF isolates information about design and structure of that legacy system using Java source code and then discover an Intermediate Model (IM) at Platform Specific Model (PSM) level. Subsequently transfigure these IMs using transformation engine into UML models consisting of both structural (UML class diagram) and behavioral models (UML activity diagram) at Platform Independent Model (PIM) level.

Applicability of Src2MoF is demonstrated using two benchmark case studies; ATM case study and Amadeus Hospitality case study.

Rest of the paper is organized as follows; **Section II** describes the literature review in the context of model driven reverse engineering for generating high level UML models from Java source code. In **Section III**, proposed methodology is described highlighting the architecture of proposed framework. The implementation detail of transformation engine using transformation rules is provided in **Section IV**. Consequently, validation of Src2MoF is performed in **Section V** using two benchmark case studies. Discussion and limitations of this research work is carried out in **Section VI.** Finally, the conclusion and future work is provided in **Section VII.**

II. LITERATURE REVIEW

With the emergence of Model Driven Engineering (MDE) in software development [2], model driven methods in reverse engineering become important. It is an active research area from the past few years where many research studies have been carried out on MDRE [3]. A concise description of most prominent approaches is given in section *II A*. Moreover, state of the art for UML structural and behavioral models from MDRE is presented in subsequent section *II B*. In section *II C*, research gap is identified.

A. MODEL DRIVEN REVERSE ENGINEERING APPROACHES

Different methods and techniques used for MDRE are explored in this section. For example, MoDisco [6] is an extensible and generic Eclipse plug-in for model driven reverse engineering proposed and implemented by H. Bruneliere et al. based on *a) model discovery and b) model understanding.* MoDisco has three layered architecture i.e. infrastructure, technologies and use case layers. It defines a basic metamodel approach for MDRE based on Knowledge Discovery Meta-model (KDM) specification to provide support for XML, JSP and Java. Model discoverer injects initial views from subject system and then transformations are performed to analyze and understand those models on higher abstraction views. MoDisco can considered a generic MDRE approach with an efficient tool support, implementing OMG standards for reverse engineering and applied on industrial case studies. However, MoDisco only deals with structural aspects and does not support the MDRE for behavioral aspects from source artefacts.

Often in RE, the only known source of knowledge is source code. To get models from this textual source, a text-to-model (T2M) set of transformations is require. A T2M transformation approach is presented by Cánovas Izquierdo and Garc ía Molina [7], defines transformation language for general purpose languages (GPLs). Authors proposed and validated a domain specific language (DSL) denominated as grammar to modeling language 'Gra2MoL'. This is a T2M transformation language which can be applied to any language conforming a grammar.

The proposed approach is evaluated practically by transforming a code written in "Delphi" into Abstract syntax model. Object Management Group (OMG) defined a standard for migrating legacy systems to new technologies known as Architecture Driven Modernization ADM [9]. ADM defines a set of standard meta-models supportive in reverse engineering under umbrella of ADM task force. Abstract Syntax Tree Meta-model (ASTM) [10], Knowledge Discovery Meta-model (KDM) [11] and Software Metrices Meta-Model (SMM) [12] are standardized meta-models used for extracting information from legacy systems. In another research, Pérez-Castillo *et al.* [11] expressed the knowledge discovery meta-model KDM in detail. The architecture, working and implementations of KDM are disclosed in this research. Likewise, objective of KDM [13] is to discover a comprehensive representation from legacy artefact is presented in this research study. ASTM [14] is used to depict programming language into an abstract syntax tree AST which is used for model driven reverse engineering. Purpose of SMM [15] is to create matrices of the system to analyze the subject system easily for software modernization.

Favre *et al.* [16] defined a process for reverse engineering of source code according to Model Driven Architecture (MDA) standards. Apart from model extraction, focus of this study is on a formal proof of models with different tools for testing and analyzing these models at unlike abstraction levels to get uniformity in reverse engineering process. In another research [17], Favre et al proposed an MDA based process for obtaining models from object-oriented code and formal techniques at meta-model level to maintain consistency in reverse engineering process. Authors anticipated combining static and dynamic analysis to get UML models from code. A framework is proposed to perform these semi-formal transformations at models and meta-models' level in this study. Furthermore, this approach aims to specify MOF meta-models and meta-model transformations in NEREUS MM language. This approach can be considered as good support to MDRE. Some advantages of this semi-automatic approach are: specification of well-known standards for defining models and meta-models i.e. UML and MOF. Consequently, assimilation of both static and dynamic analysis which can extract more aspects from legacy system and lastly, gaining the consistency in RE by formal proof of models. Authors presented a limited detail about techniques and tools used for transformation between different artefacts of legacy system.

A semi-automatic approach for MDRE is expressed in [18] by S Warwas et al. Authors intended to reverse engineer the PIM level design of BDI (Belief, Desire, Intention) form multi-agent based systems. They used a domain specific modeling language for multi-agent-systems (DSML4MAS) to depict agent's artifacts as models. RE method comprises: 1) Auto elicitation of 'Jadex model and info model' form Jadex project, 2) Remodeling of these models into Jade4PIM4Agents level models, 3) By hand refinements to indicate associations among agents. This approach is supportive in MDRE of multi agent base legacy systems. Another benefit is that this approach concerns model validation as semantics of Defined DSML are specified in Object-Z and validated through OCL constraints. Feliu Trias at al. proposed and idea towards the migration of legacy systems into modern systems in research a study [19]. An Architecture-Driven Modernization (ADM) based scheme for migration of CMS based web applications WAs (only PHP code) is proposed in this study. Abstract Syntax Tree Metamodel (ASTM) and Knowledge Discovery Metamodel (KDM) are used to view the source code in the form of models. The ADM based scheme is based on methodology of MDRE. First, T2M transformations are applied to get initial models and then restructuring is applied through M2M transformations to get higher level models. After that the new technology code is generated by transforming these models. Another approach [20] is defined by this research group for migration of legacy WAs to CMS based WAs. A toolkit *RE-CMS* is defined to automate MDRE of outdated applications to up-to-date platforms. This toolkit assists in extraction of knowledge from legacy web apps and then transforming these representations into CMS domain. Contribution of approaches defined by F. Trias et al that they expel MDRE from object-oriented domain to web applications domain. T2M transformation techniques are specified which may considered as step forward for new researchers in this field.

Rui Couto et al. defined an approach [21] for transforming the source code into models through MDRE and deduce the design pattern from these models. The main objective of the study can be summarized as:1) Create high level models (cf. PSM) from Java source code;2) Deduce design patterns on these models; 3) PSM to PIM abstraction. This approach is aided with a tool (MapIt) to perform this anticipated methodology. A semi-automatic approach for MDRE to extract business rules from Java applications is presented [22] by Normantas and Vasilecas et al. Authors intended to aid software conception and diminish maintenance expenditure by MD reverse engineering business rules out of legacy systems. First, in this approach system analysis is performed and then KDM based models are extracted i.e. Inventory model and code model. Finally, Business logic abstractions parts this code model into business logic implementations and infrastructure. As a further consideration, this generic approach can be applied on numerous enterprise systems.

We carried out a detailed overview of all the selected researches, tools, techniques their applications, empirical evaluations and future research proposals which are reported during past decade. TABLE 1 illustrates the corresponding outcomes of the prominent researches assessed in above sections. In TABLE 1, we delimit eight '8' parameters to compare the most renowned researches. 1) **Referenced Approach** column signifies the research papers(s) along with its reference. 2) The **scope** column implies the scope of MDRE approach anticipated in that paper which can be either *generic 'G'* or *specific 'S'*. Specific approach entails the

TABLE 1. Comparison of MDRE approaches.

Referenced Approach	Scope	Source Artefact	T2M Technique	Meta Models	Tools	Target Models	Type	Validation
Hugo Bruneliere et al. [6]	G	Java, JSP and XML	Model Discoverer	Java, JSP, KDM and XMI	MoDisco	Java XML *KDM*	Structural	Amadeus Hospitality (migration from VB6 to JEE)
J. Luis et al. [7]	G	Delphi Code	DSL	EBNF, ECORE and MOF	Gra2MoL *ATL* *RubyTL*	AST *CST*	Not Specified	Migration of Delphi code to Java
L. Favre et al. [17]	G	Java Code	Parser Execution Tracer	NEREUS MOF and UML	*MoDisco* *ATL*	AST *UML* *OCL*	Structural	N/A
S. Warwas et al. [18]	S	JadeX Project	DSML	JadeX and PIM4Agents	*MoDisco* *QVT*	Java Agents	Structural	JadeX Mars World Classic
F. Trias et al. [19]	S	PHP code	Parser	ASTM & KDM	RE-CMS *ATL*	AST of PHP	Structural	Websana
Normantas et al. [22]	G	Enterprise System	Extraction and discovery	KDM	*ATL* *Eclispe*	*KDM* *Java, UI*	Structural	Commercial-off-the-Shelf Enterprise Content Management
L. Martinez et al. [23]	S	Java Code	Parser	UML and KDM	*MoDisco*	UML *XML*	Behavioral	eLib Program
A Bergmayr et al. [24]	S	Java Code	Java Model Discoverer	Java and fUML	fREX	UML	Behavioral	Java Program

correspondence of MDRE to a limited technology, problem domain or case-study, while a generic approach contrariwise infers the definition of a new MDRE context or anticipating the methods/techniques in MDRE which assisted in extensive ambition irrespective of a single technology or domain. 3) In **source artefact** column, the targeted input of the proposed approach is represented. Input may be some programming language code, database or other legacy technology. 4) **T2M Technique** column describes the adopted reverse engineering method to get modeling descriptions from source antique. To represent the textual input graphically, approaches used some type of discoverers or open source parsers. Each of those approaches which are discussed or used in relevant research study are inspected. 5) **Meta-Models**: Adopted meta-models by MDRE approaches are enlisted in this column. 6) Column labeled as '**Tools**' discloses the tools used for MDRE. Some researches intended and executed a tool for MDRE specified in normal fonts i.e. H. Bruneliere et al developed MoDisco and most of the researches used certain co-existed tool e.g. L. Favre et al, enlisted in *italics*.7) The attributes under **Target Model** outcome represents that every single MDRE approach is some resultant model which is achieved after performing MDRE. There are two types of models are considered, a) Whichever proposed as the output of approach/framework/tool specified in normal font and b) whichever used anywhere between the whole process as intermediate models are shown in *italics*. 8) In column **Type**, stipulation on the type of evaluation fulfilled to expressed MDRE approach is presented. The assessment can either for inspection of static structure of the subject system 'Structural' or exploration of dynamic behaviour of subject system 'Behavioural'. 9) '**Validation**' indicates whether the anticipated approach is justified by means of some case study or industrial application or not.

B. MDRE AND UML DIAGRAMS

From existing literature about MDRE, we found some approaches aimed at generation of UML models from source artefacts. UML class diagram, component diagram, use case diagram, activity, sequence, and state machine diagram are proposed outcomes of researchers for depicting interface and workflow of the source technology. An overview of such approaches for MDRE of structural and behavioural diagrams is respectively given in subsequent sections.

1) MDRE AND UML STRUCTURAL DIAGRAMS

MoDisco [6] targeted at showing output as Java or JSP model. Java model can also be depicted in UML model

TABLE 2. Investigation of UML structural and behavioral studies using MDRE.

UML Structural Diagrams	Class diagram	[6] [16] [17] [21] [25]
	Use Case diagram	[1]
UML Behavioral Diagrams	Sequence diagram	[23]
	State Machine Diagram	[24]
	Activity diagram	[24][25]

editor as UML class diagram. Moreover, so far MoDisco is the only contribution for MDRE at industry level, so this tool has been used by researchers as for the implementation of their proposed idea. Favre *et al.* [16] defined a process for reverse engineering of source code according to MDA standards. Authors aimed to recover UML models (i.e. Class diagram) from Java code by combining static and dynamic analysis. Similarly, Pereira *et al.* [1] presented a study about recovery of Use Case diagrams from Java code by means of MDRE spotlighting on conversions at model and MM levels. To obtain MDA models through RE, old static and dynamic analysis methods are incorporated along with formal specifications by transformations at MOF meta-model level. These transformations are itemized as OCL constraints. Besides this proposal there is no validation of this approach is observed.

Semantic web assists in automation of discovery, invocation and integration of web-services by enriching them with semantic descriptions. In a research study [25], Djamel et al. proposed a model driven approach to assemble OWL-s specifications from web services. In their approach authors anticipated to use MDRE to get UML models from WSDL (Web semantics Description Language) documents and then transform these models into OWL-S specs by using different ontologies. In RE phase, authors introduce a UML profile to build representations from WSDL. UML class diagram and UML activity diagram are proposed result of this phase for modelling of interface and workflow of WSDL respectively.

2) MDRE AND UML BEHAVIOURAL DIAGRAMS

In [23] L. Martinez et al. anticipated an approach for MDRE of UML sequence diagram from Java source code using ADM standards and MoDisco platform. Methodology of MoDisco approach is used to discover Abstract models. Authors anticipated to enrich the information extraction process i.e. KDM implemented in MoDisco to extract knowledge about recovering sequence diagram from Java source. This research also aimed to contribute in MoDisco community by enriching MoDisco for MDRE of behavioral diagrams. However, there is no such evidence found in MoDisco community and this tool still lacks the MDRE of behavioral aspects from source artefact. Benefit of this approach is that this study can be considered as a good step for new research studies about MDRE of sequence diagrams from Java source code.

Bergmayr *et al.* [24] proposed a framework to RE the executable behavior of source code. The framework known as fREX aims to give support to extract model for behavior embedded in Java code. The proposed framework mainly relies on the use of Foundational UML(FUML) as a pivot language for representing application behavior.

Djamel *et al.* [25], proposed a model driven approach to assemble OWL-s specifications from web services. In their approach authors anticipated to use MDRE to get UML models (Activity Diagram) from WSDL to specify the behaviour of analysed system. II-C illustrates the relevant research studies which have focused on model driven reverse engineering to obtain structural or behavioural UML diagrams from source artefacts.

C. RESEARCH GAP

We carried out a detailed overview of all the selected researches, tools, techniques, their applications, empirical evaluations and future research proposals which are reported during past decade. **TABLE 1** illustrates the corresponding outcomes of the prominent researches assessed in above sections. We further narrow down our scope on MDRE of UML diagrams from legacy systems. II-C illustrates such researches along type of UML model supposed to be outcome of that approach. Case study used and its appliance on analysis of system is observed. We found no evidence about the experimental study of Pereira *et al.* [1], Favre *et al.* [16], and Favre *et al.* [17]. Remaining approaches justified their aim well for evaluation of static structure. Three approaches claimed the MDRE of behavioral diagrams but there is no evidence found for experimental evaluation of dynamic behavior of source artefact(s). There had been a big focus of researchers on the analysis of the structural aspects of the system but too less work is done on the RE the behavior of the system and this area is getting intentions of researchers. Consequently, there is a big research gap on modeling the behavior of a legacy system from available source artefact(s). MDRE is challenged by:

- Limited Tool Support to extract and model structural aspects of legacy system.
- Almost no support to extract and model behavioral aspects of legacy system.
- There is no such framework, which provides the support for the generation of both structural and behavioral models from Legacy system.

There is a need of complete MDRE process which manipulated source artefacts structurally as well as functionally as a whole. Our proposed approach uses the integration of both (Parser and Discoverer) accompanied by a Text-to-Model (T2M) transformation technique based on unified modeling language (UML2). The objective of this proposal is to implement an MDRE framework which extracts information from legacy systems to figure out intermediate representations (IR) and then convert these IRs into UML models. The outcome of the proposed approach is generation of high-level UML model showing structure as well as behavior, which is justified through experiment, formerly not done by any approach.

III. SOURCE-TO-MODEL FRAMEWORK (SRC2MOF)

In this section, we have developed a complete, open source Model Driven Reverse Engineering (MDRE) framework named as SRC2MOF in Eclipse facilitating extraction of UML structural and behavioral models from Java source code. Src2Mof intends to provide the capabilities for creating, analysis and use of these high-level UML models in software development. Architecture of Src2MoF is described in 'subsequent section A' providing the details of layout and components involved.

A. ARCHITECTURE OF SOURCE TO MODEL FRAMEWORK

Source to Model Framework (Src2MoF) enables the user to get the modeler view of the legacy application with all its structural aspects in class diagram, as well as the behavior embedded in Java code into modeler view as an activity diagram. Architecture of Src2MoF is shown in **FIGURE 2**. It contains a user interface, intermediate model discoverer (IMD) and model generator.

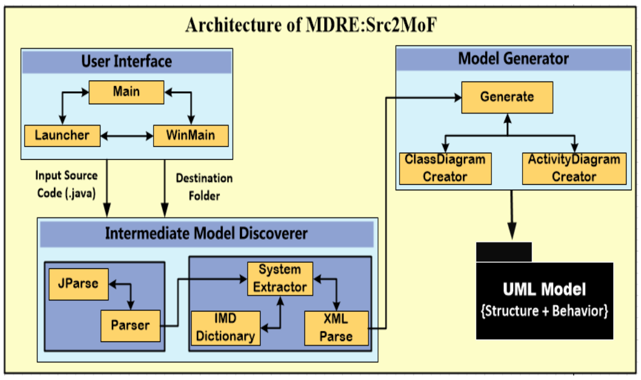


FIGURE 2. Architecture of proposed framework.

User interacts with interface; input (source code) goes to the IMD where a polished version of Abstract Syntax Tree (AST) is processed and intermediate model is obtained. We integrated source code parser in way that it creates an AST instead of parsed tree from code. AST is a polished version of parsed tree in which irrelevant implementation information of subject system, is removed and a tree is formed which contains only necessary information. Consequently, we require the structural and behavioral aspects from source code to model structure and behavior of subject system, so we implement an information extractor and intermediate discoverer within IMD. Extractor pulls out required aspects from each class of source code and passes it to discoverer where an intermediate model of subject system is formed. After this step, model generator transforms this intermediate model into UML structural and behavioral models by using predefined mapping rules. In our case we defined two templates i.e. *Class Diagram Creator and Activity Diagram Creator* inside model generator. Class diagram creator transforms structure related parts of IM into a UML class diagram and Activity diagram creator aids Generator to model behavioral aspects into a UML activity diagram. Details of implementation is given in subsequent sections.

1) USER INTERFACE

The user interface UI of our framework as illustrated in **FIGURE 2** executes the Main class. UI is the place where interactions between users and framework occur.

The objective of this interaction is to allow effective operation and control of the framework from the user end. Generally, a good designed interface makes it easy, efficient, and user-friendly to operate a system in the way which produces the desired result. Keeping in view the importance of good designed UI and consequences of poor designed interface, our focus was to build a user friendly and simple UI. The interaction between human and our framework is kept simple and we have given the maximum information on UI.

2) INTERMEDIATE MODEL DISCOVERER (IMD)

It is one of the major components of Src2MoF as illustrated in **FIGURE 2**. Implementation of this module is carried out in Java language using eclipse IDE and mapping rules are implemented for transforming the textual model into a UML model. IMD implements classes like source code parser, IMD dictionary and XMLparse. Source code parser uses an open source tool *'Java Parser'*[1]. Main function of this parser is to symbolize the source code into an abstract syntax tree, a structure that represents the code in such way, which is easy to process. This parser typically analyzes the source into a series of expressive tokens. This sequence is then passed through semantic analysis in which tokens are scanned and a parsed tree in formed. This parsed tree contains a lot of information regarding tokens position, line number, column number and unique id which is not used by our framework. In our framework, we integrated this parser as a part of intermediate model discoverer (IMD) **FIGURE 4**, such a way that it parses the source code and passes an abstract syntax tree AST to model discovery phase. Implemented IMD is generic and language independent discoverer able to support the constructs of any language to extract an intermediate model of source code.

In IMD dictionary **FIGURE 3**, information about extraction of features from parsed tree is stored. Inside IM discoverer, some meta-classes are implemented to extract features from source code. For example, *MetaDataMemeber* class is used to extract collection of attributes/properties of all classes present in source code. Similarly, *MetaMethod* class is used to extract collection of operations and *MetaFunction* class is used for extracting function-behavior.

[1]https://github.com/javaparser/javaparser

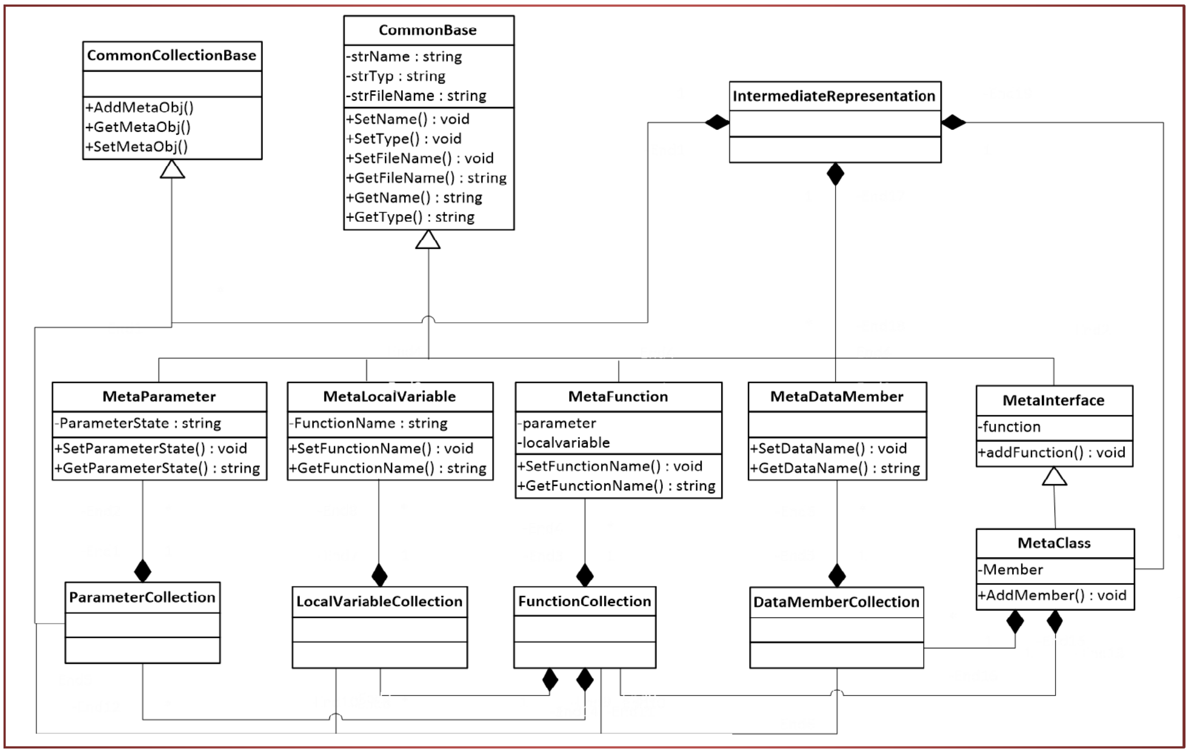
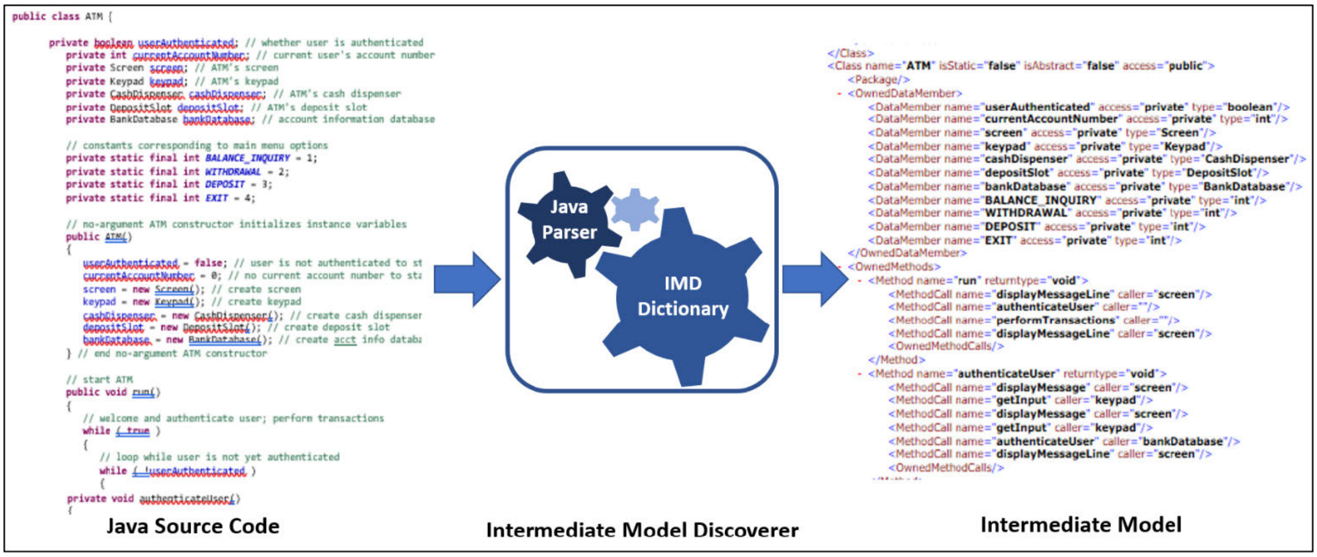


FIGURE 3. Depiction of IMD dictionary.


```
public class ATM {

        private boolean userAuthenticated; // whether user is authenticated
           private int currentAccountNumber; // current user's account number
           private Screen screen; // ATM's screen
           private Keypad keypad; // ATM's keypad
           private CashDispenser cashDispenser; // ATM's cash dispenser
           private DepositSlot depositSlot; // ATM's deposit slot
           private BankDatabase bankDatabase; // account information database

           // constants corresponding to main menu options
           private static final int BALANCE_INQUIRY = 1;
           private static final int WITHDRAWAL = 2;
           private static final int DEPOSIT = 3;
           private static final int EXIT = 4;

           // no-argument ATM constructor initializes instance variables
           public ATM()
           {
              userAuthenticated = false; // user is not authenticated to st
              currentAccountNumber = 0; // no current account number to sta
              screen = new Screen(); // create screen
              keypad = new Keypad(); // create keypad
              cashDispenser = new CashDispenser(); // create cash dispenser
              depositSlot = new DepositSlot(); // create deposit slot
              bankDatabase = new BankDatabase(); // create acct info databa
           } // end no-argument ATM constructor

           // start ATM
           public void run()
           {
              // welcome and authenticate user; perform transactions
              while ( true )
              {
                 // loop while user is not yet authenticated
                 while ( !userAuthenticated )
                 {
           private void authenticateUser()
           {
```

```
</Class>
<Class name="ATM" isStatic="false" isAbstract="false" access="public">
   <Package/>
 - <OwnedDataMember>
      <DataMember name="userAuthenticated" access="private" type="boolean"/>
      <DataMember name="currentAccountNumber" access="private" type="int"/>
      <DataMember name="screen" access="private" type="Screen"/>
      <DataMember name="keypad" access="private" type="Keypad"/>
      <DataMember name="cashDispenser" access="private" type="CashDispenser"/>
      <DataMember name="depositSlot" access="private" type="DepositSlot"/>
      <DataMember name="bankDatabase" access="private" type="BankDatabase"/>
      <DataMember name="BALANCE_INQUIRY" access="private" type="int"/>
      <DataMember name="WITHDRAWAL" access="private" type="int"/>
      <DataMember name="DEPOSIT" access="private" type="int"/>
      <DataMember name="EXIT" access="private" type="int"/>
   </OwnedDataMember>
 - <OwnedMethods>
    - <Method name="run" returntype="void">
         <MethodCall name="displayMessageLine" caller="screen"/>
         <MethodCall name="authenticateUser" caller=""/>
         <MethodCall name="performTransactions" caller=""/>
         <MethodCall name="displayMessageLine" caller="screen"/>
         <OwnedMethodCalls/>
      </Method>
    - <Method name="authenticateUser" returntype="void">
         <MethodCall name="displayMessage" caller="screen"/>
         <MethodCall name="getInput" caller="keypad"/>
         <MethodCall name="displayMessage" caller="screen"/>
         <MethodCall name="getInput" caller="keypad"/>
         <MethodCall name="authenticateUser" caller="bankDatabase"/>
         <MethodCall name="displayMessageLine" caller="screen"/>
         <OwnedMethodCalls/>
```

FIGURE 4. Input and output of IMD.

Extracted information about each class is stored in a *MetaClass* and all *MetaClasses* are combined to get intermediate representation. This intermediate representation is an XML like depiction of source code.

3) MODEL GENERATOR

The intermediate model obtained from IMD is passed to the model generator which acts like a transformation engine. Model generator implements the rules to transform intermediate models from IMD to UML models. Model generator calls two templates i.e. class diagram creator and activity diagram creator to create these models. Output of UML model generator is a UML package which encompasses a class diagram representing design and activity diagrams showing operations and functions as implemented in Java source code. Implementation of this model generator is carried out using Eclipse tool.

Src2MoF is developed as a generic, reliable and an open source MDRE framework. A pure separation of concerns is addressed in our MDRE approach. Src2MoF combines the generalized use of MDE principles with reverse engineering approaches to provide a complete and extensible solution to MDRE challenges as discussed in **section 2**. User interface of this framework is simple and user friendly. One can easily select input and output files destinations with simple *Browse* buttons. Other features of interface are self-explanatory i.e. open output file, reset and generate buttons. Similarly, a status bar is there for showing the status of output files i.e. successfully generated or some errors found. The source code of Src2MoF along with self-explanatory user manual and sample case studies are available at [34].

IV. IMPLEMENTATION

This section defines transformation rules for translating Java source code to UML models. These are two types of transformations, 1) Transformation from source code (text) to Intermediate model (textual model) and 2) transformation of intermediate model to UML models. Rules for mapping source code to intermediate models are written in Java and implemented through a source code parser. On the other hand, transformation rules to implement UML model generator are written and implemented in UML2 platform.

A. TRANSFORMATION RULES FOR TRANSLATING SOURCE-CODE TO INTERMEDIATE-MODEL-DISCOVERER

Transformation rules for transforming source code into IMD are depicted in **TABLE 3**. Glimpse of outcome for each rule is also given in the table. Each rule is also briefly described in table. Translation of structural aspects of Java code into intermediate model are easy to understand, as each class transformed into a declaration block exactly with the name of class along its access modifiers. Attributes of a class are transformed into data-members with their return types.

Behavioral information contained in Java is translated into IM as *method declaration*. We enriched IMD to correctly extract behavioral information from existing code. It extracts each method name along its return type and method call (executable behavior) with the class needs to be executed as its *caller.* **FIGURE 4** presents graphical illustration of Intermedia Model Discover. The IMD with the help of IMD dictionary generates a machine interpretable representation of Java code using source code parser.

B. TRANSFORMATION RULES FOR IMD TO UML MODEL GENERATOR

TABLE 4 illustrates the mapping of intermediate model with UML compliant model. IMD column shows parts of system so as descripted in intermediate model discoverer. UML Model column describes the program's components in UML diagram. By using given transformation rules, the intermediate models are transformed into resultant UML models representing the structure and behavior of the subject system. Class diagram describes structure in an easy way where information about structural artefacts i.e. attributes, methods, primitive types, and relationship among different entities are presented in a single diagram. Class diagram is designed to correctly express single system descriptions and the general structure of a system [31]. As described in **TABLE 4**, declaration blocks, data-members and methods are translated into corresponding UML classes, attributes and operations. Dynamic behavior of operation is transformed into *function behavior* in the same language in which it is written. UML Model generator takes function behavior containing the complete information of operation and creates a UML Activity model to illustrate behavior of models. Among all UML behavioral diagrams, the sequences of activities essential to carry out a functional goal from start to finish can be graphically represented using activity diagrams [32]. Activity diagrams appear to be more appropriate for capturing behavioral aspects in a way that they can be executed directly at model-level. Description of structural and behavioral diagrams created as an output of proposed Src2MoF is given in Validation section of this article.

From the operational viewpoint, a method written in Java is represented as operation in UML. A description of mapping of dynamic behavior between source code and UML model given in **TABLE 5**. An activity is created against each method with its formal activity nodes.

Execution of transformation rules presented in **TABLE 3**, **TABLE 4** & **TABLE 5** is done through a sequence of operations. Transformation rules are easy to understand. However, an algorithm presented **FIGURE 5** explains the **TABLE 5** more precisely. In this algorithm a process of modeling the extracted behavioral aspects in intermediate model to UML model is presented.

For each operation in Java, and activity with the name of that operation is created in UML with syntax *Act_operation name.* Return type of operation is transformed into *Activity output pin and object flow.* An *Object Action* is created for instance declaration of operation and *Call Operation Action* to show Operation Invocation method. Operation's body is translated into *Opaque Action* in activity model.

C. OVERVIEW OF TOOLS AND TECHNOLOGIES

Tools and technologies used to develop Src2MoF are presented in this section.

1) TOOLS

Eclipse and Source code parser are used to carry out implementation of Src2Mof. Eclipse [26] is an Integrated Development Environment (IDE), widely used for developing Java-based applications. It also supports other languages such as C, C++, Python, C#, COBOL.

Eclipse provides rich client platforms e.g. Workbench and SWT for development of general-purpose applications. It is a de-facto standard platform to facilitate the modeling and transformations in Model Driven Engineering (MDE). In Src2MoF, Eclipse tool is used, which facilitates the Intermediate Model Discoverer (IMD) implementation and UML2 modeling. Our model generator is using Text-to-model transformations which is also implemented in Eclipse. Source code parser is an open source tool developed in Java.

TABLE 3. Source code to intermediate model (IM) mapping rules.

Source code	IMD
Class.Java	Meta class
Description: Each java class in source code is transformed to a Meta-Class in IMD. Each meta-class contains complete info defined by common-base.	

Class Declaration
`public class Account {`
`public abstract class Transaction`
	Declaration Block with class name and access
	`<Class name="Account" isAbstract="false" access="public">`
	`<Class name="Transaction" isAbstract="true" access="public">`

Description: Declaration in source code is mapped into a declaration defined in IMD. "IsAbstract" is true for Abstract Super Class and access modifier is shown in end i.e. "access" is Public/Private etc.	

Instance variable declaration
`private int accountNumber:`
`private int pin;`
`private double availableBalance;`
`private double totalBalance;`
	Owned Data Member with same name as in source code.
	`<OwnedDataMember>`
	`<DataMember name="accountNumber" access="private" type="int"/>`
	`<DataMember name="pin" access="private" type="int"/>`
	`<DataMember name="availableBalance" access="private" type="double"/>`
	`<DataMember name="totalBalance" access="private" type="double"/>`

Description: Instances (attributes) of a class are represented by data-member in IM. Type indicates the **data type** of the variables.	
Instance of other class as variable `private Screen screen;`	Data member with type reference class `<DataMember name="screen" access="private" type="Screen"/>`
Description: Instances of referenced classes like association relationships are also shown as *Data-Members* in IM, but typed to class to which they are referenced.	

Operations
`// returns available balance`
`public double getAvailableBalance()`
`{`
`   return availableBalance;`
`} // end getAvailableBalance`
	Owned Method
	`<Method name="getAvailableBalance" returntype="double">`

Description: Operation expression in source code are mapped into *Method* in IM. Name and return type of operation are same as operation signature.	

Operation body
`public void run()`
`{`
`   screen.displayMessageLine`
`   authenticateUser();`
`} // end while`
`performTransactions();`
`userAuthenticated = false;`
`currentAccountNumber = 0;`
`screen.displayMessageLine`
	Method Call
	`- <Method name="authenticateUser" returntype="void">`
	`   <MetaLocalVariable name="accountNumber" type="int"/>`
	`   <MethodCall name="displayMessage" caller="screen"/>`
	`   <MethodCall name="getInput" caller="keypad"/>`

Description: Operation's body statement in source code is represented by different elements in IM such as Method Call, Local Variable, Conditional Statement. Statements of operation body are captured to translate them into Function-behavior using activity diagram.	

We have implemented source code parser [27] as a part of our IMD as it transforms source code in easy to process form.

2) LANGUAGES

Java language and Unified modeling languages (UML) are used to develop Src2MoF framework. Java [28] is a multi-paradigm language developed by Oracle Corporation. It is simple, robust and object oriented. We have used Java language for the development of Intermediate Model Discoverer (IMD) and source code parser used in Src2MoF.

UML2 [29] is an Eclipse Modeling Framework (EMF) based implementation of UML 2.x for eclipse. It contains two main categories to generalize diagrams i.e. Structural diagrams and Behavioral diagrams. We have used UML2 to implement the UML model generator in our framework. Transformation templates i.e. class diagram creator and activity diagram creator are written for UML2. In other words, high-level UML models from intermediate model are generated using UML2.

V. VALIDATION

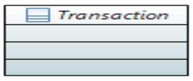
The validation is done through software engineering experts, who tested this framework on 5 benchmark case studies. The strategy used to evaluate the reliability of tests for proposed framework consists of two steps. 1) First, experts have manually performed modeling (designed models) of case studies using class diagram and activity diagram by means of different software design tools i.e. Eclipse, papyrus, starUML, rationalrose, 2) Second, same case studies are applied to the Src2MoF (proposed framework). A comparison is performed by the output of our framework and manually designed models by design experts. Two of those benchmark case studies

TABLE 4. Transformation rules for intermediate model (IM) to UML model.

IMD	UML Model	Description

Intermediate Representation
`<?xml version="1.0" encoding="UTF-8"?>`
`- <JavaClassInfo xmlns="./file.xml">`
	UML package
	platform:/resource/OutputModel/ClassDiagram.uml
	<Package>
		Collection of meta-classes formed during IMD process, translated to UML package.

Meta-Class collection
`+ <Class name="Account"`
`+ <Class name="ATM"`
`+ <Class name="ATMCaseStudy"`
	UML Classes with names.
	<Package>
	<Class> Account
	<Class> ATM
	<Class> ATMCaseStudy
		Every declaration block transforms to a Class with specified name.

Abstract Class `<Class name="Transaction" isAbstract="true"`	UML Class (Name in *Italics*) *<Class> Transaction*  	A class where "Is Abstract" is "true" is mapped into an abstract super class and its name is in *Italics.*
Access Modifier `<Class name="Account" access="public">`	Visibility <Class> Account Visibility Public	Access Modifiers of a class in IM is transformed to visibility in UML diagram.

Owned Data Member
`- <OwnedDataMember>`
`<DataMember name="pin"`
`<DataMember name="input"`
	Attributes (With name in 2nd compartment)
	<Property> pin :
	<Property> input :
		Each data member in IMD is transformed to the attribute/property in class diagram

Data member showing reference
`<DataMember name="screen" type="Screen"/>`
`<DataMember name="bankDatabase" type="BankDatabase"/>`
	Association
	<Property> screen : Screen
	<Property> bankDatabase : BankDatabase
	<Association> ATMScreen
	<Association> ATMBankDatabase
		Data member, whose return type is an object name, is transform to association with that object.

Method `<Method name="execute"`	Operation with specified name. <Operation> execute () : Name execute Namespace <Class> Deposit	Every *method* is mapped into an operation along its name and its specified class.

are given in this article; 1) Automated Teller Machine (ATM) case study and 2) Amadeus Hospitality case study.

A. ATM CASE STUDY

In this section, the complete description of the selected functionality of the ATM is provided. Total of eight classes are implemented in Java source code to design an ATM system. The description of each class is as follows:

ATM:

This class depicts and manages the overall ATM system, its operations and all ATM parts. It is required to perform the 'Startup operation, run the ATM, servicing customers, until the switch is turned off.

ATM Classes:

Classes like *keypad, screen, cash-dispenser and deposit-slot* builds the parts of an ATM Machine. Keypad.Java class signifies the keyboard from which user inputs the information to ATM machine. This class has one attribute and has one method *getInput* to return an integer value input by user. Class screen.Java denotes the screen of ATM machine. Output and messages are shown on screen. Screen.Java class has one method *displaymessage* to show the message, message line or currency amount to the customer. Cashdispenser class signifies the part of ATM which consist of cash and is responsible for dispensing cash. Two methods in this class, i.e. *dispensecash* checks whether *IsSufficientCashAvailable* and then returns the message. Class *DepositSlot* represents the ATM's deposit slot. This class has no properties and only one method *isEnvelopeReceived* which directs whether a deposit envelope was received or not.

Class Account:

This section declares the class that represents in the customer's account in bank, implementing the corresponding use case. An *account* class consists of account number of users, its PIN and information about balance in that account. It performs the operation of validating the entered PIN. There are methods for returning balance information and debit/credit in *Account* class.

TABLE 5. Description of mapping behavioral aspects between java and UML activity.

Java Concept	IMD Concept	UML Concept

Operation	Method (name = operation name)
	`</Method>`
	`- <Method name="authenticateUser" returntype="void">`
		<Function Behavior> B_operaction_name
		<Function Behavior> B_authenticateUser

Operation body	Declaration Block
	`- <Method name="authenticateUser" returntype="void">`
	`<MetaLocalVariable name="accountNumber" type="int"/>`
	`<MethodCall name="displayMessage" caller="screen"/>`
	`<MethodCall name="getInput" caller="keypad"/>`
	`<MethodCall name="displayMessage" caller="screen"/>`
	`<MethodCall name="getInput" caller="keypad"/>`
	`<MethodCall name="authenticateUser" caller="bankDatabase"/>`
	`<MethodCall name="displayMessageLine" caller="screen"/>`
	`<OwnedMethodCalls/>`
	`</Method>`
		<Activity> Act_operation_name
		<Activity> Act_authenticateUser

		<Initial Node> ●—
		<Final Node> →◉
Return type	Return type `<Method name="run" returntype="void">`	<Output Pin> <Output Pin> [0..1] <Literal Integer> pin
		<Object Flow> Object Flow

Operation's parameters	List of parameters
	`- <OwnedMethods>`
	`- <Method name="validatePIN" returntype="boolean">`
	`<MetaParameter name="userPIN" type="int"/>`
	`<OwnedMethodCalls/>`
	`</Method>`
		<Input Pin>
		<Input Pin>
		<Literal String> "\nEnter your PIN: "

		<Object Flow> Object Flow

Operation method Invocation	Method Call
	`MethodCall name="displayMessage" caller="screen"`
	`MethodCall name="getInput" caller="keypad"/>`
		Call Operation Action
		<Call Operation Action> CallOperation

Variable/ Instance creation	`<MetaLocalVariable name="accountNumber" type="int"/>`	Create Object Action <Create Object Action> userAccount
Execution Block	Individual steps	Opaque Action <Opaque Action> userAccount.validatePIN(userPIN)

Class Transaction:

This abstract superclass expresses the display of an ATM transaction. It comprises the common features of subclasses Balance-Inquiry, Withdrawal and Deposit. *Transaction* class has three public get methods *getAccountNumber, get-Screen* and *getBankDatabase. Transaction* class inherits subclasses and gives them access to class Transaction's private attributes.

Class Bank Database:

This class describes and implements the database of the bank in which information i.e. account number, PIN, Balance about customer's account(s) is saved. *BankDataBase* consists of an array of accounts. It takes the account number of the user, authenticates it and returned the complete information about that account i.e. Balance, Transactions record, account number, PIN.

Class Balance Inquiry:

Class BalanceInquiry extends Transaction and represents a balance-inquiry ATM transaction. BalanceInquiry does not have any attributes of its own, but it inherits *Transaction's* attributes *account-Number*, *screen* and *bank-Database*, which are accessible through Transaction's public, get methods.

Class ATM Case study:

Class *ATMCaseStudy* is the main class that allows us to start or "turn on," the ATM and tests the implementation of system.

RESULTS

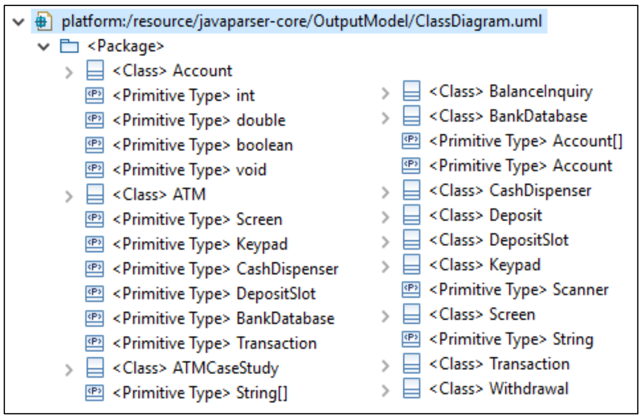
The classes of ATM case study are written using Java programming language. The input of ATM case study as Java project is provided to the Src2MoF. Generated output of the framework is in UML model editor as depicted in **FIGURE 6** and represented in Papyrus model editor in **FIGURE 7**.

a: CLASS DIAGRAM

By opening the generated output file, we get UML class diagram. Generated UML class diagram is depicted in **FIGURE 6** in UML model editor. We select UML class diagram to describe structure of analyzed legacy system. Class diagram describes structure in an easy way where information about structural artefacts i.e. attributes, methods, primitive types, and relationship among different entities are presented in a single model. In **FIGURE 6**, all ATM classes i.e. *Account, ATM, Screen, Keypad* are present as generated

```
for all JavaPackages as pkg in project do
    create umlPackage as umlPkg with pkg.name
    for all JavaClasses jClass in pkg do
        create umlClass as umlCls in umlPkg
        for all DataMembers jData in jClass do
            if type of jData is primitive then
                create uml PrimitiveType with jData.type
                create uml Property with type pt in umlCls
            end if
        end for
        for all Methods jMethod in jClass do
            for all Parameters mPara in jMethod do
                create list of args with mPara.name
                create list of argType with mPara.type
            end for
            create return Type as return with jMethod.return
            create uml Operation (List<args>, List<argType>, return) ops in umlCls
            create uml OwnedBehavior for uml ops
            for all Statements stmt in jMethod do
                create uml Activity Ac named as jMethod.name
                create initialNode
                LastActivityNode = initialNode
                if stmt is a return statmemt then
                    create ActivityNode OpaqueAtion
                    Create ActivityEdge (OpaqueAtion) <- (LastActivityNode)
                end if
                if stmt is if-else structure then
                    create ActivityNode ConditionalNode for if statmemt
                    if if-else statmemt is not empty then
                        create LocalPrecondition for ConditionalNode
                    end if
                    create ActivityEdge (ConditionalNode) <- LastActivityNode
                end if
                if stmt is method call or expression then
                    create ActivityNode CallOperationAct
                    create ActivityEdge (CallOperationAct) <- LastActivityNode
                end if
                if stmt is Local Varible then
                    create ActivityNode ObjectActionNode
                    create ActivityEdge (ObjectActionNode) <-
                    LastActivityNode
                end if
            end for
            create ActivityFinalNode
            create ActivityEdge (ActivityFinalNode) <- LastActivityNode
        end for
        for all DataMembers and Local Varibles not of primitive type do
            create Association relationship between container and contained class
        end for
        if baseclass tag is set true for jClass then
            create Generalization (jClass) <- jClass.Parent
        end if
```

FIGURE 5. Pseudo code for UML model generation algorithm.



```
platform:/resource/javaparser-core/OutputModel/ClassDiagram.uml
  <Package>
    <Class> Account
    <Primitive Type> int
    <Primitive Type> double
    <Primitive Type> boolean
    <Primitive Type> void
    <Class> ATM
    <Primitive Type> Screen
    <Primitive Type> Keypad
    <Primitive Type> CashDispenser
    <Primitive Type> DepositSlot
    <Primitive Type> BankDatabase
    <Primitive Type> Transaction
    <Class> ATMCaseStudy
    <Primitive Type> String[]
    <Class> BalanceInquiry
    <Class> BankDatabase
    <Primitive Type> Account[]
    <Primitive Type> Account
    <Class> CashDispenser
    <Class> Deposit
    <Class> DepositSlot
    <Class> Keypad
    <Primitive Type> Scanner
    <Class> Screen
    <Primitive Type> String
    <Class> Transaction
    <Class> Withdrawal
```

FIGURE 6. Output model in UML model editor.

by Src2MoF. We can see that, there are primitive types in output model which shows information about return types for attributes of each class.

Internal structural aspects of every single class can be seen by expanding that class as shown in **FIGURE 8**. There are *properties* representing *attributes* of class and *Operations* for describing functionality of that class. Function of each operations is extracted and described in *Function Behavior*. Therefore, we can conclude that, structure of ATM system has been successfully generated by Src2MoF.

The information regarding a class can be checked by expanding that class(s). We have expanded only one class Account as shown in the **FIGURE 8**. By opening the generated model in papyrus modeling editor where, a class diagram is generated with name in first compartment, properties are placed in second compartment, operations and nested classifiers are allocated to third and fourth compartment respectively.

It can be seen that, Account class has four attributes and six operations, which are same as in source code.

b: ACTIVITY DIAGRAM

MDRE of behavioral aspects of existing applications (source code) is the main area of focus in our research. Many UML models i.e. sequence, state-machine and activity diagrams are being used to represent the functionality of system. We have

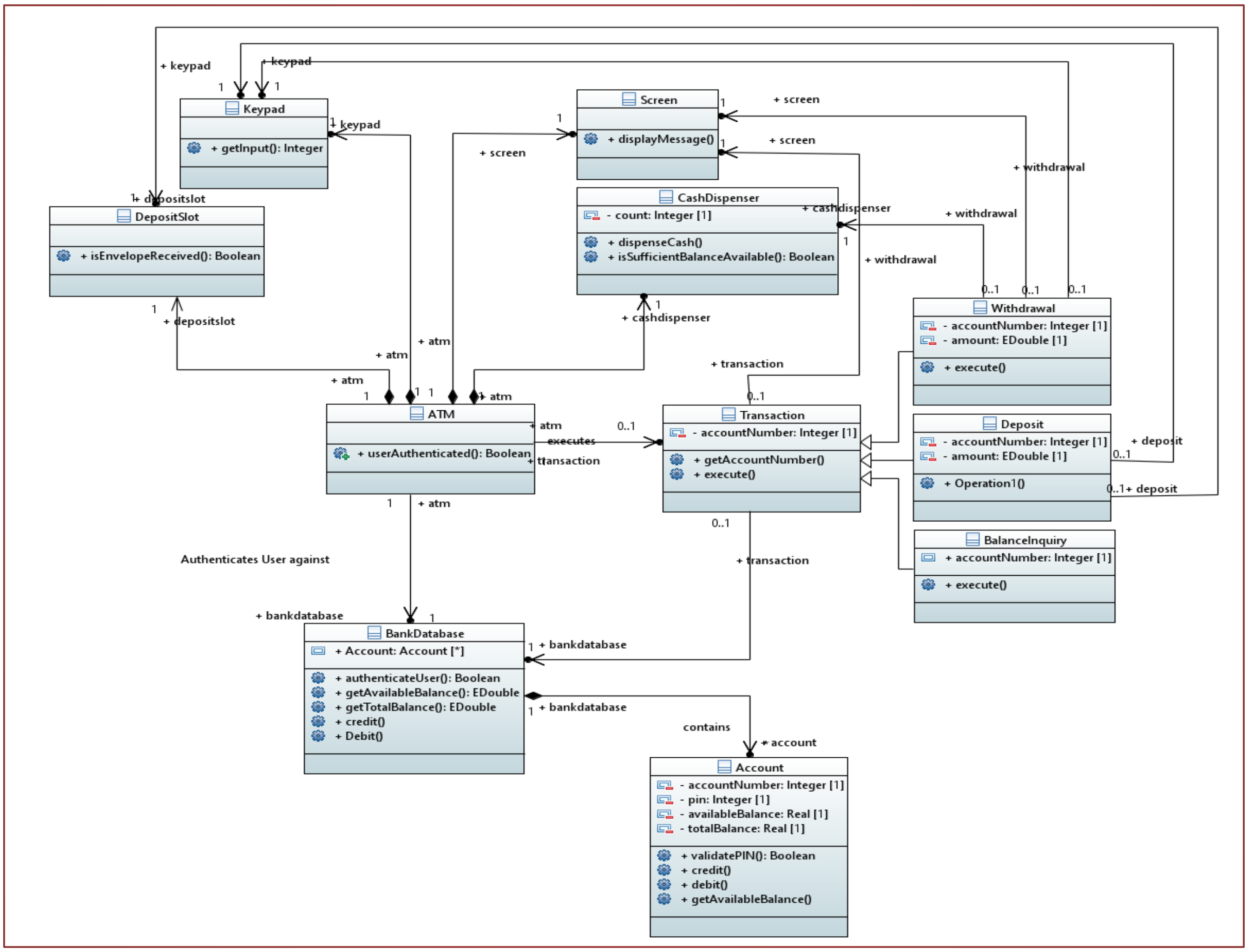
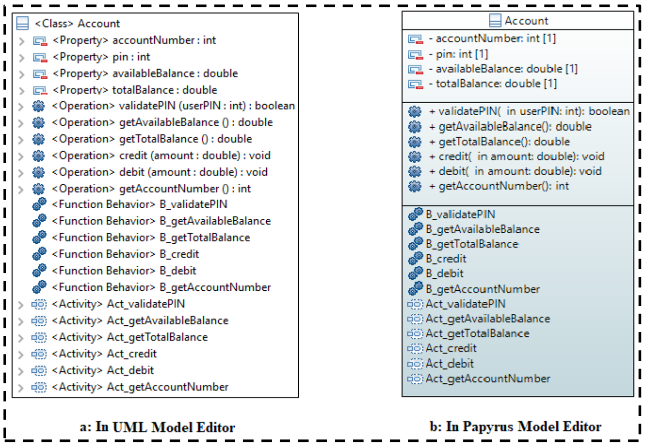


FIGURE 7. Generated UML class mode.


<Class> Account
<Property> accountNumber : int
<Property> pin : int
<Property> availableBalance : double
<Property> totalBalance : double
<Operation> validatePIN (userPIN : int) : boolean
<Operation> getAvailableBalance () : double
<Operation> getTotalBalance () : double
<Operation> credit (amount : double) : void
<Operation> debit (amount : double) : void
<Operation> getAccountNumber () : int
<Function Behavior> B_validatePIN
<Function Behavior> B_getAvailableBalance
<Function Behavior> B_getTotalBalance
<Function Behavior> B_credit
<Function Behavior> B_debit
<Function Behavior> B_getAccountNumber
<Activity> Act_validatePIN
<Activity> Act_getAvailableBalance
<Activity> Act_getTotalBalance
<Activity> Act_credit
<Activity> Act_debit
<Activity> Act_getAccountNumber

Account
- accountNumber: int [1]
- pin: int [1]
- availableBalance: double [1]
- totalBalance: double [1]
+ validatePIN(in userPIN: int): boolean
+ getAvailableBalance(): double
+ getTotalBalance(): double
+ credit(in amount: double): void
+ debit(in amount: double): void
+ getAccountNumber(): int
B_validatePIN
B_getAvailableBalance
B_getTotalBalance
B_credit
B_debit
B_getAccountNumber
Act_validatePIN
Act_getAvailableBalance
Act_getTotalBalance
Act_credit
Act_debit
Act_getAccountNumber

a: In UML Model Editor

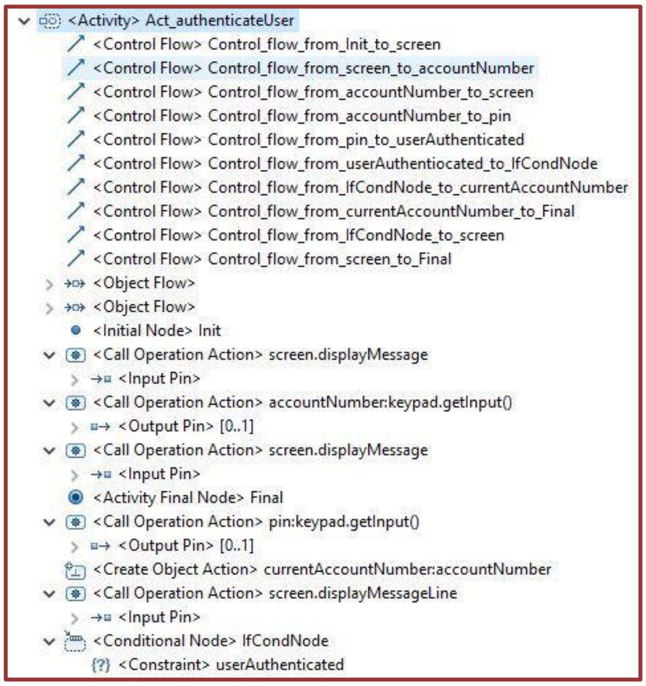
b: In Papyrus Model Editor

FIGURE 8. UML model of account class.

selected activity diagram to illustrate behavior of system. Among all UML behavioral diagrams, the sequences of activities essential to carry out a functional goal from start to finish can be graphically represented using activity diagrams [32]. Moreover, activity models have many options like conditional nodes and loops to describe a function i.e. if-else statement and switch statements. In **FIGURE 9** and **FIGURE 10**, the generated activity model for function *AuthenticateUser* of class ATM are shown. The same activity diagram is depicted in two separate notations. Representation of activity models in UML model editor (**FIGURE 9**) does not keep the constructs of activity model in order. Therefore, graphical representation along with structural and source code artefacts is illustrated in **FIGURE 10**.

We have almost extracted all the aspects related to behavior of source code and collected them to form an activity model. Our framework has created all the necessary aspects to show the behavior of the system in different modeling notation. In UML classes operations inside classes has some function behavior with a return caller. This behavior from source code is mapped into a nested classifier 'Function Behavior'. An activity diagram against each function behavior is formed which shows the behavior embedded in source code using modeling notation.

In this article, we illustrate the generated activity diagram for another operation '*validatePIN*' of class *Account* in



```
<Activity> Act_authenticateUser
    <Control Flow> Control_flow_from_Init_to_screen
    <Control Flow> Control_flow_from_screen_to_accountNumber
    <Control Flow> Control_flow_from_accountNumber_to_screen
    <Control Flow> Control_flow_from_accountNumber_to_pin
    <Control Flow> Control_flow_from_pin_to_userAuthenticated
    <Control Flow> Control_flow_from_userAuthentiocated_to_IfCondNode
    <Control Flow> Control_flow_from_IfCondNode_to_currentAccountNumber
    <Control Flow> Control_flow_from_currentAccountNumber_to_Final
    <Control Flow> Control_flow_from_IfCondNode_to_screen
    <Control Flow> Control_flow_from_screen_to_Final
  > <Object Flow>
  > <Object Flow>
    <Initial Node> Init
  v <Call Operation Action> screen.displayMessage
      > <Input Pin>
  v <Call Operation Action> accountNumber:keypad.getInput()
      > <Output Pin> [0..1]
  v <Call Operation Action> screen.displayMessage
      > <Input Pin>
    <Activity Final Node> Final
  v <Call Operation Action> pin:keypad.getInput()
      > <Output Pin> [0..1]
    <Create Object Action> currentAccountNumber:accountNumber
  v <Call Operation Action> screen.displayMessageLine
      > <Input Pin>
  v <Conditional Node> IfCondNode
        <Constraint> userAuthenticated
```

FIGURE 9. Activity model for operation "Authenticate User".

FIGURE 11. Each function behavior contains behavior which conforms source code. Transformation of *operation* in Java code to *function behavior* and *activity model* is graphically represented in **FIGURE 11 (a, b and c)** respectively.

c: DESCRIPTION OF BEHAVIORAL MODEL GENERATION

Operation inside UML class diagram is transformed to *function behavior* and then for each function behavior an activity model is generated which consist of sequence of operations represented as nodes and edges. The behavior of each method is completely described by the generated activity model and the aspects of activities are extracted and modeled according to mapping rules (**TABLE 5**).

Overview of different artifacts and models generated by Src2MoF framework given a piece of java code is shown in **FIGURE 9**. Notably, java source code and its corresponding structural and behavior model in UML Editor and in the form of activity diagram is presented. Structural aspects (e.g. ATM class, Property) are depicted through UML Class elements and behavior is illustrated using UML Activity.

To express the dynamic behavior of source code graphically, an activity model is generated against each operation with operation's name inside class model as a nested classifier. As an activity explicitly defines control nodes at which the execution starts and ends when it is invoked, those nodes, i.e., InitialNode and FinalNode, are created by default for each activity. A method invocation is mapped to UML as *CallOperationAction* for operation calling. Respective *Input pins and Object Flows* are also created with *CallOperationAction* for passing values to invoked methods and for showing the target of invoked method respectively. An instance variable is mapped in UML as *CreateObjectAction.* Additionally, an output pin is also generated to pass the object at runtime.

There are *Conditional Nodes* to represent 'if else statements' inside source code. The constraint inside conditional node describes the condition for an action to be true or false. There are two types of edges, control flow edge and object flow edge (**FIGURE 9**). Control flow edge represents the flow of activity and mostly generated by following sequential flow of source code or logical order in which elements has to appear in activity diagram. Control flow indicates one atomic statement or function call is executed and control is transferred to another activity node. While the object flow indicates the distribution of object to different nodes. The same object can be possibly distributed to several actions. The object flow connects the object to activity nodes by action input pin and output pin. Object flow edge not necessarily follow the control flow edge as objects may have different life and scope than the nodes of an activity. Object flow edges can be used to model input parameters of function (see userPIN input pin **FIGURE 11 (c)**), object instantiation, parameter passing and return values through input and output pins. The source code of Src2MoF along with user manual and ATM case study is available at [34] for further evaluation.

B. AMADEUS HOSPITATLITY CASE STUDY

Amadeus [33] provides services of reservation and information about Flights, hotels, stadiums, casinos and other locations for travelers and tourists. Amadeus helps in technology modernization, which facilitates its clients at every step of the journey.

In other words, it is cloud-based technology solutions for global hospitality organizations.

1) DESCRIPTION

In this section an overview of the classes is given. Amadeus package comprises of seven classes and few APIs.

a) Amadeus: This class initializes the API client. Comprises of three fields/attributes, two public methods and two inherited methods.

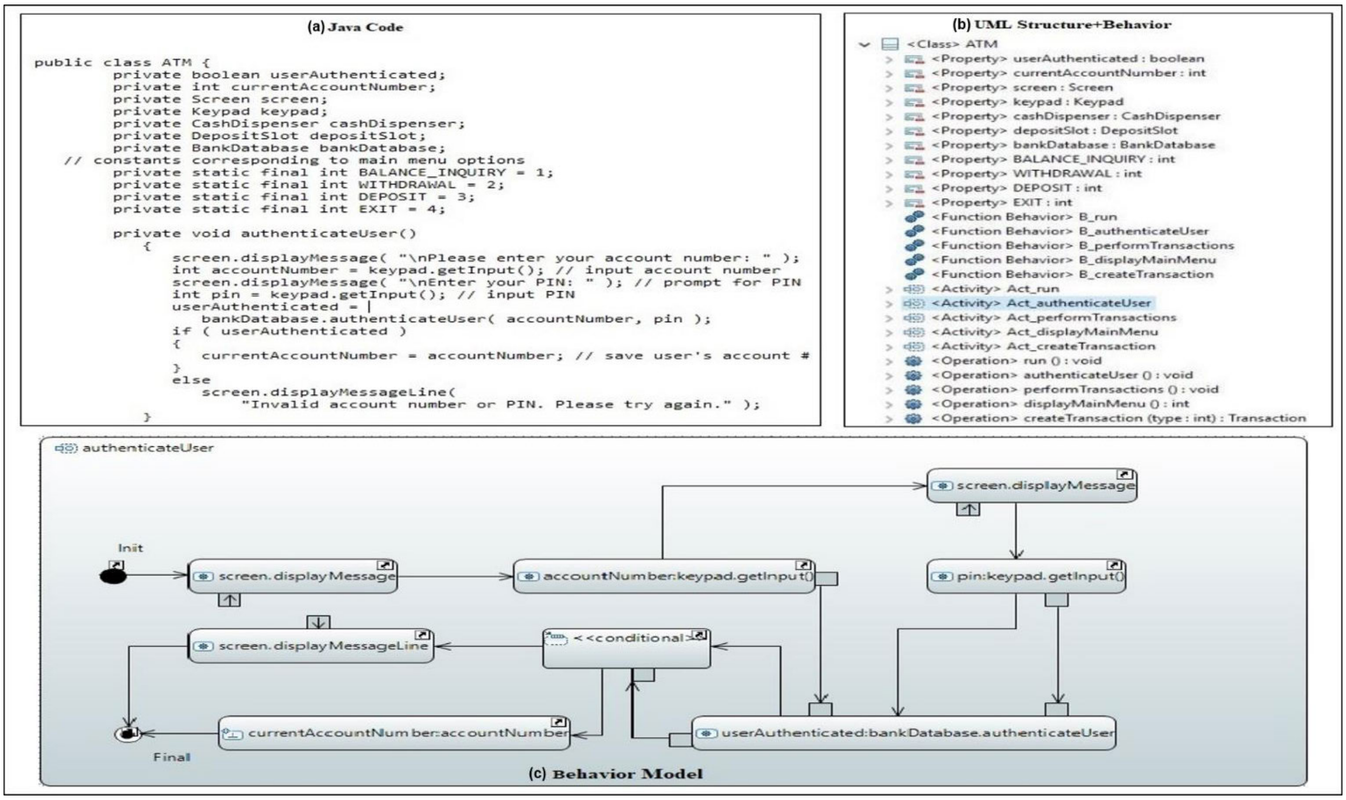
b) Configuration: This class shows the configuration of Amadeus API client. It has no attributes and 20 public methods.

c) Constants: This class expresses all constant variables used to implement this system.

d) HTTPClient: This class shows Amadeus API clients, http part. It consists Resource, Response and Configuration public methods.

2) RESULTS

In Src2MoF, input of Amadeus case study is browsed for Java input project and destination folders is selected. Consequently, generate button is clicked. Status bar shows "Successfully generated UML files".


(a) Java Code

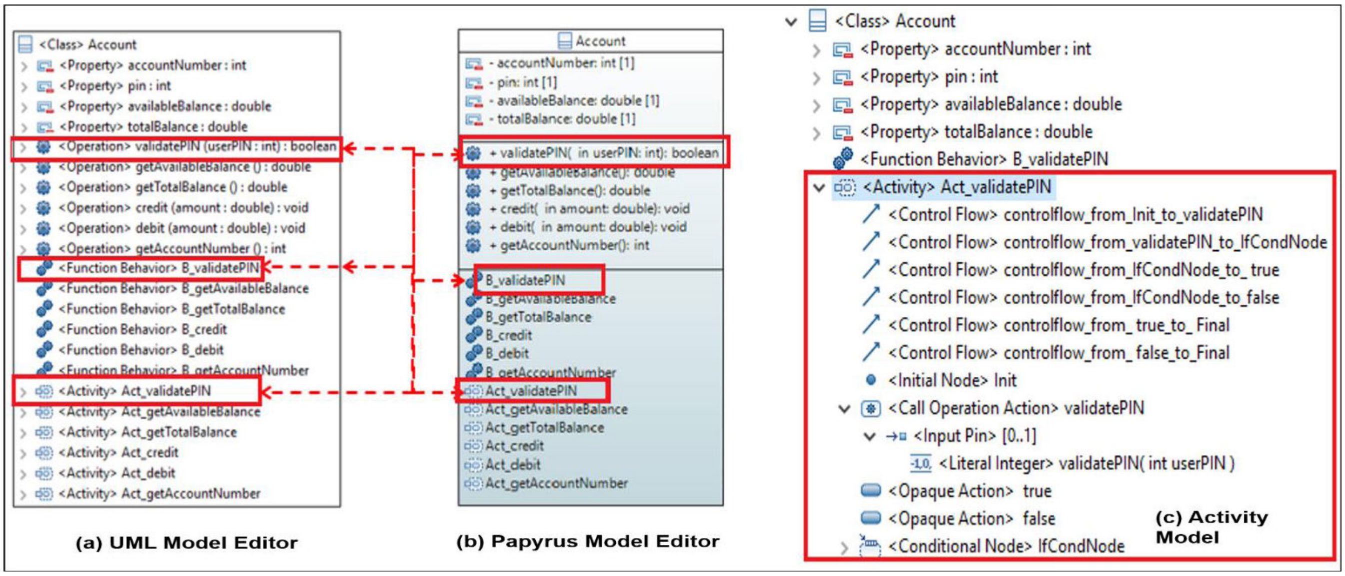
```
public class ATM {
        private boolean userAuthenticated;
        private int currentAccountNumber;
        private Screen screen;
        private Keypad keypad;
        private CashDispenser cashDispenser;
        private DepositSlot depositSlot;
        private BankDatabase bankDatabase;
   // constants corresponding to main menu options
        private static final int BALANCE_INQUIRY = 1;
        private static final int WITHDRAWAL = 2;
        private static final int DEPOSIT = 3;
        private static final int EXIT = 4;

        private void authenticateUser()
           {
              screen.displayMessage( "\nPlease enter your account number: " );
              int accountNumber = keypad.getInput(); // input account number
              screen.displayMessage( "\nEnter your PIN: " ); // prompt for PIN
              int pin = keypad.getInput(); // input PIN
              userAuthenticated = |
                 bankDatabase.authenticateUser( accountNumber, pin );
              if ( userAuthenticated )
              {
                 currentAccountNumber = accountNumber; // save user's account #
              }
              else
                 screen.displayMessageLine(
                      "Invalid account number or PIN. Please try again." );
           }
```

(b) UML Structure+Behavior

```
<Class> ATM
  <Property> userAuthenticated : boolean
  <Property> currentAccountNumber : int
  <Property> screen : Screen
  <Property> keypad : Keypad
  <Property> cashDispenser : CashDispenser
  <Property> depositSlot : DepositSlot
  <Property> bankDatabase : BankDatabase
  <Property> BALANCE_INQUIRY : int
  <Property> WITHDRAWAL : int
  <Property> DEPOSIT : int
  <Property> EXIT : int
  <Function Behavior> B_run
  <Function Behavior> B_authenticateUser
  <Function Behavior> B_performTransactions
  <Function Behavior> B_displayMainMenu
  <Function Behavior> B_createTransaction
  <Activity> Act_run
  <Activity> Act_authenticateUser
  <Activity> Act_performTransactions
  <Activity> Act_displayMainMenu
  <Activity> Act_createTransaction
  <Operation> run () : void
  <Operation> authenticateUser () : void
  <Operation> performTransactions () : void
  <Operation> displayMainMenu () : int
  <Operation> createTransaction (type : int) : Transaction
```

authenticateUser
Init
screen.displayMessage
accountNumber:keypad.getInput()
screen.displayMessage
pin:keypad.getInput()
screen.displayMessageLine
<<conditional>>
currentAccountNumber:accountNumber
userAuthenticated:bankDatabase.authenticateUser
Final
(c) Behavior Model

FIGURE 10. Activity model for operation "Authenticate User" in Papyrus Editor.


(a) UML Model Editor

```
<Class> Account
  <Property> accountNumber : int
  <Property> pin : int
  <Property> availableBalance : double
  <Property> totalBalance : double
  <Operation> validatePIN (userPIN : int) : boolean
  <Operation> getAvailableBalance () : double
  <Operation> getTotalBalance () : double
  <Operation> credit (amount : double) : void
  <Operation> debit (amount : double) : void
  <Operation> getAccountNumber () : int
  <Function Behavior> B_validatePIN
  <Function Behavior> B_getAvailableBalance
  <Function Behavior> B_getTotalBalance
  <Function Behavior> B_credit
  <Function Behavior> B_debit
  <Function Behavior> B_getAccountNumber
  <Activity> Act_validatePIN
  <Activity> Act_getAvailableBalance
  <Activity> Act_getTotalBalance
  <Activity> Act_credit
  <Activity> Act_debit
  <Activity> Act_getAccountNumber
```

(b) Papyrus Model Editor

```
Account
- accountNumber: int [1]
- pin: int [1]
- availableBalance: double [1]
- totalBalance: double [1]
+ validatePIN( in userPIN: int): boolean
+ getAvailableBalance(): double
+ getTotalBalance(): double
+ credit( in amount: double): void
+ debit( in amount: double): void
+ getAccountNumber(): int
B_validatePIN
B_getAvailableBalance
B_getTotalBalance
B_credit
B_debit
B_getAccountNumber
Act_validatePIN
Act_getAvailableBalance
Act_getTotalBalance
Act_credit
Act_debit
Act_getAccountNumber
```

(c) Activity Model

```
<Class> Account
  <Property> accountNumber : int
  <Property> pin : int
  <Property> availableBalance : double
  <Property> totalBalance : double
  <Function Behavior> B_validatePIN
  <Activity> Act_validatePIN
    <Control Flow> controlflow_from_Init_to_validatePIN
    <Control Flow> controlflow_from_validatePIN_to_IfCondNode
    <Control Flow> controlflow_from_IfCondNode_to_true
    <Control Flow> controlflow_from_IfCondNode_to_false
    <Control Flow> controlflow_from_true_to_Final
    <Control Flow> controlflow_from_false_to_Final
    <Initial Node> Init
    <Call Operation Action> validatePIN
      <Input Pin> [0..1]
        <Literal Integer> validatePIN( int userPIN )
    <Opaque Action> true
    <Opaque Action> false
    <Conditional Node> IfCondNode
```

FIGURE 11. Detailed depiction of how behavior is mapped to activity (for operation *ValidatePIN*).

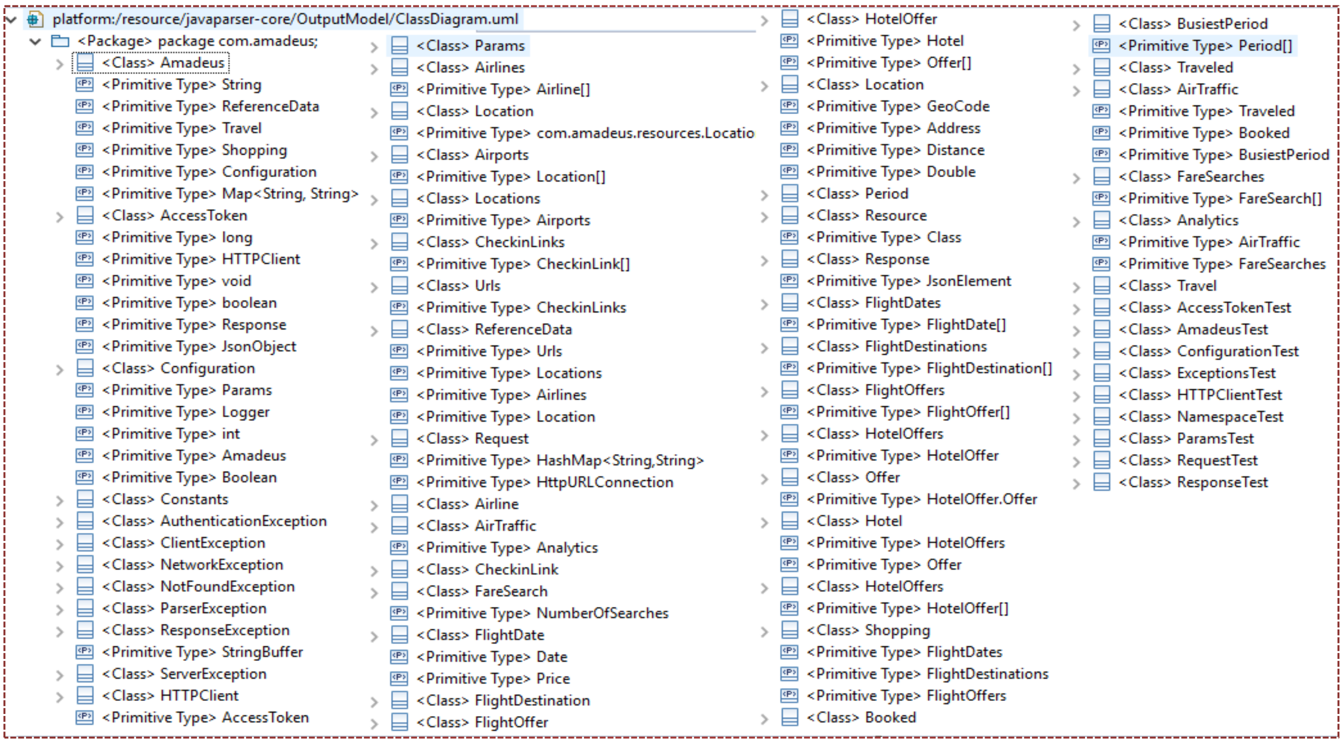
a: CLASS DIAGRAM

Output model is shown in **FIGURE 12**. A list of classes as well as exceptions are generated as present in source code. Attributes of each class have some return type value i.e. int. string etc. Similarly, associations of classes have also return types with the name of that associated e.g. < Primitive Type > *Shopping* and *Travel*. Return types of each class are presented as "Primitive Types" appended.

b: ACTIVITY DIAGRAM

We expand one class (**FIGURE 13**), Amadeus to illustrate its properties, operations, and function behavior with corresponding activity model for that function behavior.

The relationship of superclass and child class concept as used in source code, is mapped to generalization in class diagram. < Generalization > (see HTTPclient) element maps the parent and child relationship between Amadeus and



```
platform:/resource/javaparser-core/OutputModel/ClassDiagram.uml
  <Package> package com.amadeus;
    <Class> Amadeus
    <Primitive Type> String
    <Primitive Type> ReferenceData
    <Primitive Type> Travel
    <Primitive Type> Shopping
    <Primitive Type> Configuration
    <Primitive Type> Map<String, String>
    <Class> AccessToken
    <Primitive Type> long
    <Primitive Type> HTTPClient
    <Primitive Type> void
    <Primitive Type> boolean
    <Primitive Type> Response
    <Primitive Type> JsonObject
    <Class> Configuration
    <Primitive Type> Params
    <Primitive Type> Logger
    <Primitive Type> int
    <Primitive Type> Amadeus
    <Primitive Type> Boolean
    <Class> Constants
    <Class> AuthenticationException
    <Class> ClientException
    <Class> NetworkException
    <Class> NotFoundException
    <Class> ParserException
    <Class> ResponseException
    <Primitive Type> StringBuffer
    <Class> ServerException
    <Class> HTTPClient
    <Primitive Type> AccessToken
    <Class> Params
    <Class> Airlines
    <Primitive Type> Airline[]
    <Class> Location
    <Primitive Type> com.amadeus.resources.Locatio
    <Class> Airports
    <Primitive Type> Location[]
    <Class> Locations
    <Primitive Type> Airports
    <Class> CheckinLinks
    <Primitive Type> CheckinLink[]
    <Class> Urls
    <Primitive Type> CheckinLinks
    <Class> ReferenceData
    <Primitive Type> Urls
    <Primitive Type> Locations
    <Primitive Type> Airlines
    <Primitive Type> Location
    <Class> Request
    <Primitive Type> HashMap<String,String>
    <Primitive Type> HttpURLConnection
    <Class> Airline
    <Class> AirTraffic
    <Primitive Type> Analytics
    <Class> CheckinLink
    <Class> FareSearch
    <Primitive Type> NumberOfSearches
    <Class> FlightDate
    <Primitive Type> Date
    <Primitive Type> Price
    <Class> FlightDestination
    <Class> FlightOffer
    <Class> HotelOffer
    <Primitive Type> Hotel
    <Primitive Type> Offer[]
    <Class> Location
    <Primitive Type> GeoCode
    <Primitive Type> Address
    <Primitive Type> Distance
    <Primitive Type> Double
    <Class> Period
    <Class> Resource
    <Primitive Type> Class
    <Class> Response
    <Primitive Type> JsonElement
    <Class> FlightDates
    <Primitive Type> FlightDate[]
    <Class> FlightDestinations
    <Primitive Type> FlightDestination[]
    <Class> FlightOffers
    <Primitive Type> FlightOffer[]
    <Class> HotelOffers
    <Primitive Type> HotelOffer
    <Class> Offer
    <Primitive Type> HotelOffer.Offer
    <Class> Hotel
    <Primitive Type> HotelOffers
    <Primitive Type> Offer
    <Class> HotelOffers
    <Primitive Type> HotelOffer[]
    <Class> Shopping
    <Primitive Type> FlightDates
    <Primitive Type> FlightDestinations
    <Primitive Type> FlightOffers
    <Class> Booked
    <Class> BusiestPeriod
    <Primitive Type> Period[]
    <Class> Traveled
    <Class> AirTraffic
    <Primitive Type> Traveled
    <Primitive Type> Booked
    <Primitive Type> BusiestPeriod
    <Class> FareSearches
    <Primitive Type> FareSearch[]
    <Class> Analytics
    <Primitive Type> AirTraffic
    <Primitive Type> FareSearches
    <Class> Travel
    <Class> AccessTokenTest
    <Class> AmadeusTest
    <Class> ConfigurationTest
    <Class> ExceptionsTest
    <Class> HTTPClientTest
    <Class> NamespaceTest
    <Class> ParamsTest
    <Class> RequestTest
    <Class> ResponseTest
```

FIGURE 12. Output Model of Amadeus Program.

TABLE 6. Comparative analysis.

MDRE Approaches	Input	Targeted Output Model	Target Model Type	Tool	Contribution to MDRE
H. Bruneliere et al. [6]	Java Code XML	KDM Model Java Model	Structural	MoDisco	Partial
S. Warwas et al. [18]	JadeX project	Java Model	Structural	Nil	Partial
F. Trias et al [20]	PHP code	AST Model	Structural	RE-CMS	Partial
Normantas et al. [22]	Enterprise Project	KDM Model Java Model	Structural	Nil	Partial
A. Bergmyr et al. [24]	Java Code	UML Model	Behavioral	fREX	Partial
Proposed Approach	Java Project	UML Model	**Structural** **Behavioral**	Src2MoF	Full

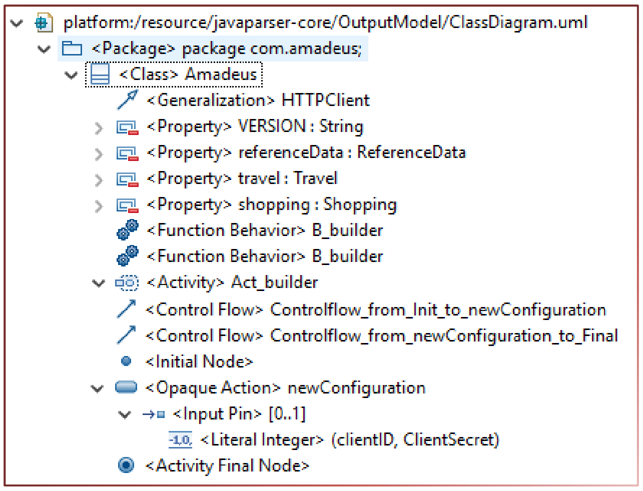
HTTPclient classes. Data members and operations of class Amadeus are translated into < Property > and < Function Behavior > respectively.

After checking results of all five case studies, it can be analyzed that Src2MoF framework can correctly generate the structure as well as behavior UML models together from Java source code. Generated artefacts are industry standard models, which conforms UML class diagram and UML activity diagram meta-models.

VI. COMPARATIVE ANALYSIS

We have proposed Src2MoF to support model driven reverse engineering. This framework is a complete MDRE solution for reverse engineering of the structural as well as behavioural aspects of legacy subject system. In this section, we compare Src2MoF with renowned state-of-the-art approaches in the area of MDRE. We utilized five evaluation parameters for this comparison i.e. *Input artefact, proposed Output, type of analysis (structure or behaviour), Tools (proposed/used) and Contribution in the field of MDRE on the basis of experimental validation of approach.*

TABLE 6 gives a description of this comparison. In MDRE approaches [6], [18], [20], [22], researchers focus on MDRE of structures of source artefacts only. As we defined earlier, the complete or full MDRE approaches comprise of modeling



```
platform:/resource/javaparser-core/OutputModel/ClassDiagram.uml
  <Package> package com.amadeus;
    <Class> Amadeus
      <Generalization> HTTPClient
      <Property> VERSION : String
      <Property> referenceData : ReferenceData
      <Property> travel : Travel
      <Property> shopping : Shopping
      <Function Behavior> B_builder
      <Function Behavior> B_builder
      <Activity> Act_builder
        <Control Flow> Controlflow_from_Init_to_newConfiguration
        <Control Flow> Controlflow_from_newConfiguration_to_Final
        <Initial Node>
        <Opaque Action> newConfiguration
          <Input Pin> [0..1]
            <Literal Integer> (clientID, ClientSecret)
        <Activity Final Node>
```

FIGURE 13. **Amadeus activity model for operation builder.**

both structure as well as behavior from legacy system's source code are hard to find in the literature'. Therefore, we consider all these researches as ''Partial'' contribution to MDRE. Similarly, the approach at [24] discussed about MDRE of behavior embedded in Java code but apart from a single conference paper which proposes this approach, the detailed overview of tool and its implementations is completely missing. Moreover, the proposed framework is only for the MDRE of behavior, so we require some other approach also to discover structure when we are dealing with fREX. On the other hand, in Src2MoF, we implemented a complete MDRE approach which deals with the RE of both structural as well as behavioral aspects of legacy system. Therefore, it can be concluded that ''Src2MoF'' is the first attempt which gives the complete automated solution for modeling structure along with the behavior of legacy systems through Java source code.

VII. DISCUSSIONS AND LIMITATIONS

We have proposed Src2MoF to support model driven reverse engineering. This framework is a complete MDRE solution for reverse engineering of the structural as well as behavioural aspects of legacy subject system. Previously, a very few researchers explore this area, as Brunelière *et al.* [6] implemented a generic and extensible framework for MDRE so-called MODISCO. However, it is only for modeling the structure of the legacy systems and then analysis of that extracted model. Pereira *et al.* [1] also contributed to MDRE but their scope was only for structure of the legacy system using MDRE. Martinez *et al.* [23] however anticipated to get behavioral models like sequence diagram from object-oriented code but there is no realistic evaluation of their work. Similarly, Bergmayr *et al.* [24] presented a RE framework for executable behavior of source code. The framework known as fREX aims to give support to extract model of behavior embedded in Java code, but still this work is also not well validated through proper experiments or case studies. Apart from these, no well-defined and precise research regarding MDA based reverse engineering of behavioral aspects is hard to find in literature.

Therefore, we can claim that ''Src2MoF'' is the first attempt which gives the complete solution i.e. modeling structure along with behavior of source code. Furthermore, Src2MoF provides a complete tool support and its applicability is demonstrated through industrial case-studies. The following are the key features of Src2MoF:

Genericity: Src2MoF is generic approach that can be used for MDRE of any type of object-oriented legacy system. Before this, only MoDisco was available as a generic framework for MDRE but it is not a complete solution of MDRE.

Structural and behavioral modeling: Src2MoF automatically extracts the structural information from source code and model it into a UML class diagram. It also supports the reverse engineering of behavior embedded in source code and model it into an activity diagram.

Limitations: We have proposed a generic approach for MDRE, and a source code parser is assembled in ''Intermediate Model Discoverer'. We have configured this parser in a way that it makes IMD a generic and language independent, while working with language other than Java. This source code parser is an open source tool and IMD can be enriched to limit the use of source code parser. Consequently, to represent the behavior of the legacy system, UML activity model is used. In UML activity diagram, some constructs are facilitated such as operations and functions, but some nodes are not facilitated such as merge node and fork node.

VIII. CONCLUSION AND FUTURE WORK

This article presents Model Driven Reverse Engineering (MDRE) solution to generate the high-level models of both structural as well as behavioral aspects of legacy systems from Java source code. As a part of research, an open source ''**Sour**ce to **Mo**del **F**ramework (Src2MoF) is developed. Particularly, an intermediate model discoverer is developed by integrating source code parser to get the intermediate model of system's structure and behavior from the existing Java code. Second, highly level UML models comprise of Class and Activity diagrams are generated from intermediate model which represent system structure and behavior respectively. To the best of our knowledge, Src2MoF is the first attempt which gives the complete solution for MDRE i.e. modeling structure along with behavior of legacy systems from source code. The broader applicability of Src2MoF has been demonstrated through benchmark case studies.

Currently, Src2MoF only deals with the Java source code. In this regard, we intend to extend Src2MoF for other programming languages like C# etc. Furthermore, we also plan to represent the structure and behavior of system in different other UML diagrams like sequence, component etc. for detailed analysis of legacy systems.

REFERENCES

[1] P. Claudia, M. Liliana, and F. Liliana, "Recovering use case diagrams from object oriented code: An MDA-based approach," in *Proc. 8th Int. Conf. Inf. Technol. New Generations*, Apr. 2011, pp. 737–742.

[2] P. L. Ferreira, S. Hammoudi, and B. Selic, Eds., *Model-Driven Engineering and Software Development: 5th International Conference, MODELSWARD 2017, Porto, Portugal, February 19–21, 2017, Revised Selected Papers*, vol. 880. New York, NY, USA: Springer, 2018.

[3] C. Raibulet, F. A. Fontana, and M. Zanoni, "Model-driven reverse engineering approaches: A systematic literature review," *IEEE Access*, vol. 5, pp. 14516–14542, 2017.

[4] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: A generic and extensible framework for model driven reverse engineering," in *Proc. IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2010, pp. 173–174.

[5] M. W. Anwar, M. Rashid, F. Azam, and M. Kashif, "Model-based design verification for embedded systems through SVOCL: An OCL extension for SystemVerilog," *Des. Automat. For Embedded Syst.*, vol. 21, no. 1, pp. 1–36, 2017, doi: 10.1007/s10617-017-9182-z.

[6] H. Brunelière, J. Cabot, G. Dupé, and F. Madiot, "MoDisco: A model driven reverse engineering framework," *Inf. Softw. Technol.*, vol. 56, no. 8, pp. 1012–1032, 2014.

[7] J. L. C. Izquierdo and J. G. Molina, "Extracting models from source code in software modernization," *Softw. Syst. Model.*, vol. 13, no. 2, pp. 713–734, 2014.

[8] U. Sabir, F. Azam, and M. W. Anwar, "A comprehensive investigation of model driven architecture (MDA) for reverse engineering," in *Proc. Int. Conf. Softw. E-Bus.*, 2017, pp. 43–48.

[9] OMG. *Architecture Driven Modernization*. Accessed: May 2019. [Online]. Available: https://www.omg.org/adm/

[10] J. Canovas and J. G. Molina, "An architecture-driven modernization tool for calculating metrics," *IEEE Softw.*, vol. 27, no. 4, pp. 37–43, Jul./Aug. 2010.

[11] R. Pérez-Castillo, I. G.-R. de Guzmán, and M. Piattini, "Knowledge discovery metamodel-ISO/IEC 19506: A standard to modernize legacy systems," *Comput. Standards Interfaces*, vol. 33, no. 6, pp. 519–532, 2011.

[12] F. García, M. Serrano, J. Cruz-Lemus, F. Ruiz, and M. Piattini, "Managing software process measurement: A metamodel-based approach," *Inf. Sci.*, vol. 177, no. 12, pp. 2570–2586, 2007.

[13] Object Management Group. *Knowledge Discovery Metamodel (kdm)*. Accessed: May 2019. [Online]. Available: http://www.omg.org/%0Atechnology/kdm/index.htm

[14] I. Object Management Group. *Abstract Syntax Tree Metamodel (ASTM)*. Accessed: May 2019. [Online]. Available: http://www.omg.org/spec/ASTM

[15] I. Object Management Group. *Structured Metrics Meta-Model (SMM)*. Accessed: May 2019. [Online]. Available: http://www.omg.org/spec/SMM/

[16] L. Favre, L. Martinez, and C. Pereira, "MDA-Based reverse engineering of object oriented code," *Enterprise, Business-Process and Information Systems Modeling* (Lecture Notes in Business Information Processing), vol. 29. New York, NY, USA: Springer, 2009, pp. 251–263.

[17] L. Favre, "Formalizing MDA-based reverse engineering processes," in *Proc. 6th Int. Conf. Softw. Eng. Res. Manage. Appl.*, Aug. 2008, pp. 153–160.

[18] S. Warwas and M. Klusch, "Making multiagent system designs reusable: A model-driven approach," *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Intell. Agent Technol.*, vol. 2, Aug. 2011, pp. 101–108.

[19] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos, "Reverse engineering applied to CMS-based Web applications coded in PHP: A proposal of migration," in *Communications in Computer and Information Science*, vol. 417. New York, NY, USA: Springer, 2013, pp. 241–256.

[20] F. Trias, V. de Castro, M. López-Sanz, and E. Marcos, "RE-CMS: A reverse engineering toolkit for the migration to CMS-based sWeb applications," in *Proc. 30th Annu. Symp. Appl. Comput.*, 2015, pp. 810–812.

[21] R. Couto, A. N. Ribeiro, and J. C. Campos, "A patterns based reverse engineering approach for java source code," in *Proc. 35th Annu. IEEE Softw. Eng. Workshop*, Oct. 2012, pp. 140–147.

[22] O. Vasilecas and K. Normantas, "Deriving business rules from the models of existing information systems," in *Proc. 12th Int. Conf. Comput. Syst. Technol.*, 2011, pp. 95–100.

[23] L. Martinez, C. Pereira, and L. Favre, "Recovering sequence diagrams from object-oriented code: An ADM approach," in *Proc. 9th Int. Conf. Eval. Novel Approaches Softw. Eng. (ENASE)*, Apr. 2014, pp. 1–8.

[24] A. Bergmayr, H. Bruneliere, J. Cabot, J. García, T. Mayerhofer, and M. Wimmer, "FREX: FUML-based reverse engineering of executable behavior for software dynamic analysis," in *Proc. 8th Int. Workshop Modeling Softw. Eng.*, vol. 16, 2016, pp. 20–26.

[25] A. B. Djamel, B. Djamal, and M. Mimoun, "A reverse engineering approach for specifying Semantic Web Service with respect to MDA," in *Proc. 3rd Int. Conf. Inf. Commun. Technol. From Theory Appl.*, Apr. 2008, pp. 1–8.

[26] *Eclipse*. Accessed: May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Eclipse_(software)

[27] D. van Bruggen. *Java Parser*. Accessed: May 2019. [Online]. Available: http://Javaparser.org/

[28] *Java*. Accessed: May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Java_(programming_language)

[29] *UML2*. Accessed: May 2019. [Online]. Available: http://wiki.eclipse.org/MDT/UML2

[30] Harvey Deitel—Deitel&Associates. *ATM Case Study*. Accessed: May 2019. [Online]. Available: https://www.oreilly.com/library/view/Javatm-for-programmers/9780137018529/aph.html

[31] A. Kästner, M. Gogolla, and B. Selic, "From (imperfect) object diagrams to (imperfect) class diagrams: New ideas and vision paper," in *Proc. 21th ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst.*, vol. 18, 2018, pp. 13–22.

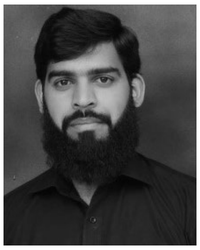
[32] G. Chong and Z. Jun, "Analysis and design of Internet-based library management system based on UML," in *Proc. 14th Int. Conf. Innov. Manage.* Cardiff, U.K.: Univ. of Wales, Sep. 2017, pp. 1077–1083.

[33] *Amadeus Hospitality*. Accessed: Dec. 2019. [Online]. Available: https://en.wikipedia.org/wiki/Amadeus_IT_Group

[34] *Source to Model Framework–Src2MoF*. Accessed: May 2019. [Online]. Available: https://ceme.nust.edu.pk/ISEGROUP/Src2MoF/index

UMAIR SABIR received the B.E. and M.S. degrees in software engineering from the Mirpur University of Science and Technology and National University of Sciences and Technology (NUST), respectively. He is currently a Research Scholar with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering (CEME), National University of Sciences and Technology (NUST), Islamabad, Pakistan. His current research interests include model driven software development and model driven reverse engineering.

FAROOQUE AZAM has been teaching various software engineering courses, since 2007. He is currently an Adjunct Faculty Member with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His areas of interests include model driven software engineering, business modeling for web applications, and business process reengineering.



SAMI UL HAQ received the B.Sc. degree in computer science from the Virtual University of Pakistan, in 2011, the M.Sc. degree in information technology from Quaid-i-Azam University, in 2013, and the M.S. degree in software engineering from the National University of Sciences and Technology, Pakistan, where he is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME. His research interest includes analysis, design, and development of model-based innovative solutions.

MUHAMMAD WASEEM ANWAR is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology, Pakistan. He is also a Senior Researcher and an Industry Practitioner in the field of Model Based System Engineering (MBSE) for embedded and control systems. His major research interest includes model based system engineering (MBSE) for complex and large systems.

WASI HAIDER BUTT is currently an Assistant Professor with the Department of Computer and Software Engineering, College of Electrical and Mechanical Engineering, National University of Sciences and Technology, Pakistan. His areas of interests include model driven software engineering, and web development, and requirement engineering.

ANAM AMJAD received the B.S. degree in computer sciences and the M.S. degree in software engineering from International Islamic University and NUST, respectively. She is currently pursuing the Ph.D. degree with the Department of Computer and Software Engineering, CEME, National University of Sciences and Technology (NUST), Pakistan. Her area of research interests includes business process automation through model driven software engineering.