Java Swing Modernization Approach
Complete Abstract Representation based on Static and Dynamic Analysis

Zineb Gotti and Samir Mbarki
MISC Laboratory, Faculty of Science, Ibn Tofail University, BP 133, Kenitra, Morocco

Keywords: Architecture-Driven Modernization (ADM), Static and Dynamic Analysis, Legacy System, Knowledge Discovery Model (KDM), Graphical User Interface Meta-Model (GUIM), Abstract Syntax Tree Meta-Model (ASTM), Java Development Tool (JDT), Reverse Engineering, Parsing, Slicing, Interaction Flow Modelling Language (IFML), Task Model, Empirical Analysis.

Abstract: GUIs are essential components for today software. However, legacy applications do not benefit from the advantages of user interfaces new technologies that enhance the interaction and the quality of the system. Building a new system from another existing one is more requested and a very complex process. So, we opted for an ADM approach based on the development of separate models capturing various aspects such as tasks, presentation and structures of system dialogue and behavior. For this purpose, the software artifacts should be analyzed and corresponding behavioral and structural models must be created. Two forms of this analysis were developed: a static analysis that provides the ability to retrieve information from the application using the source code and a dynamic analysis for extracting information about application behavior in run mode. This paper presents the automation of the extraction process, which permits understanding and analyzing the behavior of the legacy system, and compares the models generated to deduce the best solution for an abstract representation of existing GUI's models.

1 INTRODUCTION

It is necessary to migrate from old and obsolete systems to others that are new and effective, in order to follow the evolution of technology and evolve to better system engineering practices such as model-driven engineering MDE. The software modernization refers to understanding and evolving legacy softwares in order to maintain their business (Ramón et al., 2010). The Object Management Group OMG has defined an architecture-driven modernization initiative ADM (http://adm.omg.org) in 2003 to extend MDA practices and standards with existing systems. It is intended for the standard representation of reverse engineering applications.

In this work, we refer to an approach that is based on this initiative to define abstract models and automate their generation through transformation chains. These models capture knowledge related to the GUI and manipulate this knowledge to migrate from one context to another (See Figure 1). All these models will be described by meta-models, and correspondences between related models will be defined by transformation rules.

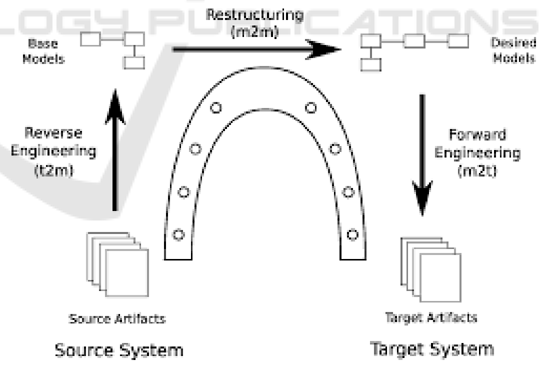


Figure 1: ADM Horseshoe model (http://adm.omg.org).

To meet the modernization requirements, ADM defines two models: ASTM and KDM. (http://adm.omg.org). These models are used to capture design knowledge required to build the future user interface (UI).

The migration process consists of two phases: (Mbarki et al., 2015).

The model discovery of the legacy system represents the extraction of information from the source code; a text to model transformation for building models code. This paper proposes a method to retrieve important elements such as the components of the graphical user interface and the relationships between these components. All the information generated in this step was incorporated into concrete PSM models which are ASTM and GUIM.

The Restructuring: it is a model to model transformation for constructing abstract models; a representation at a higher level of abstraction defined in three PIM models: KDM, IFMLM and TASKM.

The process illustrated in Figure 2, is based on the analysis of both structural and behavioral aspects of graphical user interfaces, and sophisticated algorithms of reverse engineering.

In this article, we describe the three end-generated models that are used to represent user interfaces at a higher level of abstraction. We compare the result and try to deduce the optimal solution for presenting the existing system artifacts.

The rest of this article is organized as follows: section 2 describes the process based on the ADM approach and describes their different phases. In section 3, we illustrate our proposal by different case studies. Section 4 is devoted to the analysis of the process result. Section 5 covers the related works. Finally, section 6 concludes the work and presents the perspectives.

2 PROPOSED PROCESS

The Modernization is the practice of understanding and evolving existing software to take advantage of the new technologies' benefits. It is a process to generate modern systems. In general, it includes all activities related to the improvement of software understanding and various quality parameters, such as the complexity, maintainability, and reusability. Thus, it will extend the lifetime of a software system.

OMG has defined an ADM initiative related to the construction of standards that can be applied to modernize legacy systems. This initiative develops a set of standards to facilitate interoperability between modernization tools; we focus on the KDM and ASTM in particular.

In this article, we propose an ADM based approach allowing an abstract presentation of interactive systems. We present below the reengineering process which is divided into two phases: Discovery Model and Restructuring phases.

The process allows the extraction of GUI knowledge that will be presented in ASTM models, a model expressing the syntax of the source code via an abstract syntax tree, and also in GUIM, a model representing the graphical components, their relationship and their properties. The result of the extraction is subsequently converted into three independent platform models which are KDM, IFML and TASKM.

Analytical techniques were used throughout the process: a static analysis to extract information from the source code; information on the hierarchy of user interfaces; and dynamic analysis in order to extract information in run mode, information about the behavior of graphical user interfaces.

2.1 Model Discovery

The first step is to analyze the source code of the legacy system to discover its corresponding PSM models. It defines the structures and relationships between system elements. It enables the extraction of information from the system and stores it in concrete models such as ASTM and GUIM.

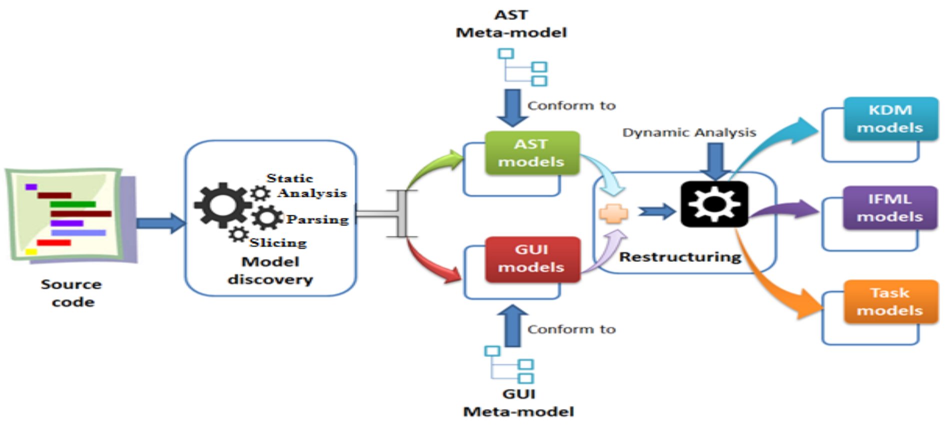


Figure 2: Overview of the migration process.

The main purpose is to analyze an interactive system developed in Java Swing. In this present stage, we opted for a static analysis to extract any information related to the syntax and structure of the Java source code as well as the presentation of graphical user interfaces.

2.1.1 Static Analysis

Its main purpose is to analyze and describe the structure of the Java source code. An ASTM model is obtained from the source code using a java development tool parser. (https://eclipse.org/jdt).

Firstly our parser compiles the Java source code to build the equivalent AST tree that will be subsequently used as a source for the next parsing.

To generate the first model, the parser calls a visitor for each AST node and creates the corresponding elements in the ASTM model respecting its meta-model (http://www.omg.org/spec/ASTM).

As depicted in Figure 3, there are two main parsing classes; ASTParser and ASTVisitor. The ASTParser class calls the class ASTVisitor to cross the various nodes of AST using the visit () method.

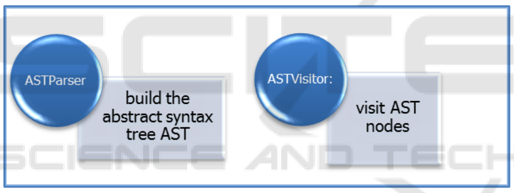


Figure 3: Parsing Classes.

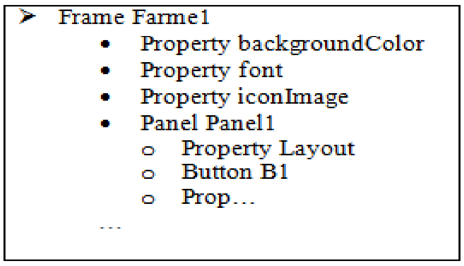
Regarding the second GUIM model representing the presentation layer; it is necessary to extract only the information related to user interfaces. A method was used that isolates the subprogram java swing from the full program which is the slicing method.

Slicing (Harman and Hierons, 2001) or the cutting of a program is a technique that allows you to define an explicit recursive function that traverses the source program AST by identifying all the fragments of programs that interact with the graphical user interface and returns the sub-tree of Swing (Silva et al., 2006). This technique allows us to ignore irrelevant details and focus only on the presentation layer.

The model GUIM result represents all the graphics components, their relationships and their properties according to GUIM meta-model (see figure 5).

There are different types of widgets (frames, buttons, text fields, labels, tables ...), and each widget is characterized by a set of graphical properties (background color, font type...), there are also the layouts used for the spatial distribution of the application elements of view.

In Figure 4 we consider the frame Frame1. It has properties such as background-Color and font type and each property has a value. This frame contains a panel to be structured. It encapsulates a set of widgets and each widget has its own properties. The panel has a layout property that is responsible for managing the location of its widgets.



- Frame Farme1
  - Property backgroundColor
  - Property font
  - Property iconImage
  - Panel Panel1
    - Property Layout
    - Button B1
    - Prop…
  - …

Figure 4: Graphical User Interface Model components.

2.2 Restructuration

This phase consists of analyzing all the information obtained from the previous phase and presents them in a higher level of abstraction.

In this present step, we develop a model-to-model QVT transformation. The entries of this transformation are the ASTM and GUIM models generated from the first step. The output is the KDM model (http://www.omg.org/spec/KDM/1.1/PDF/2009), the IFML model (http://www.ifml.org/) and the task model. (http://www.w3.org/TR/task-models).

The transformation mapping allows us to just extract the static aspect from GUIM and ASTM models, but the information related to the GUIs execution behavior in running mode is absent.

A dynamic analysis was used during this stage to deduce the behavioral aspect of the GUI system.

2.2.1 Dynamic Analysis

The main objective of our approach based on ADM is to extract both the structure and behavior of graphical interfaces implementation; in fact, user interfaces have a static part that is related to the presentation of the information and a dynamic part which is associated with the behavior. The runtime behavior is created from the execution of the graphical user interface.

The behavior is defined by events; each widget is able to trigger certain types of events under certain

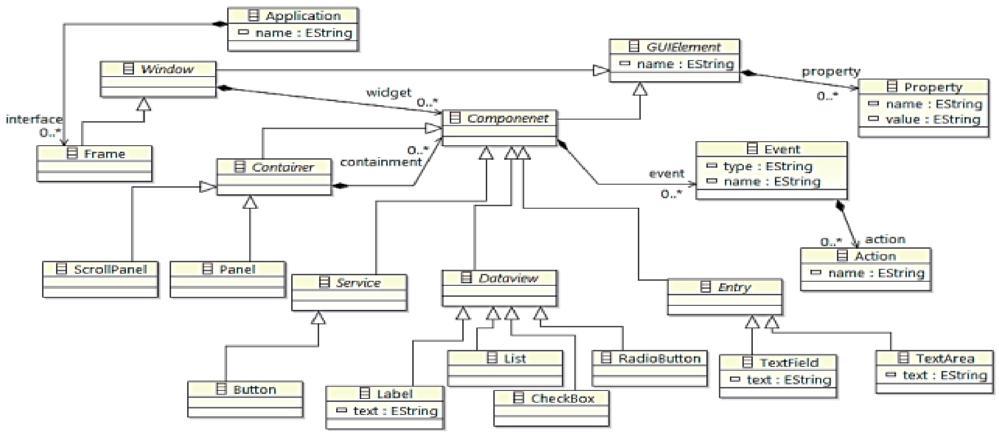


Figure 5: GUIM Meta-Model.

conditions.

Events are generated as a result of the user interaction with the user interface widgets. For example, clicking on a button, moving the mouse, entering a character, selecting an item from the list and scrolling down the page are the activities that cause events that perform actions.

During this analysis we traverse the GUIM model in order to extract all executable widgets, i.e. Widgets that trigger events and cause actions, according to the widget value we search in the ASTM model the instruction block responsible to perform the actions.

Consider the example of an action for opening a new window Window2 after a click on a button B located inside a Window1. First we travel the GUIM model corresponding to window1 to find the B button that triggered the opening. According to its value we search in the ASTM model the instruction block that presents the action to perform. We look by following on the inside of this block for the invocation method "setVisible ()" which will launch the opening of the second window. The invoker of this method is the window2.

With this analysis, we were able to deduce the dynamic relationships between the windows.

Restructuring stage identified three PIM models that describe the result of the static and dynamic analysis in a higher level of abstraction. These models are the KDM model, the IFML model and TASKM model.

KDM Model

To support modernization activities OMG defined KDM standard for the representation of existing software systems. This is a meta-model used to represent the system artifacts in a high level of abstraction. It is the basic element for the ADM approach. It provides a knowledge intermediate representation of existing software systems.

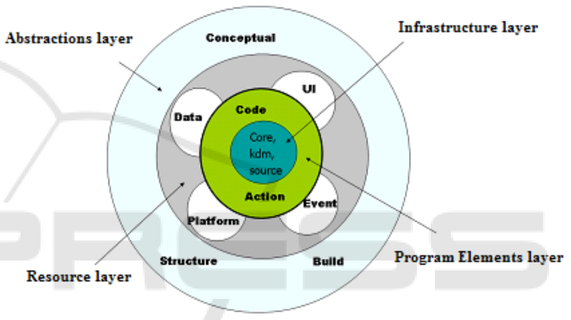


Figure 6: KDM Architecture (http://www.omg.org/spec/KDM/1.1/PDF/2009).

Figure 6 shows that KDM has twelve packages organized in four layers. From the viewpoint of the graphical user interfaces migration, four of these packages can be useful: the code, the action, the user interface and KDM packages. The package code includes the meta-model elements that represent program elements, such as data types, data elements, classes, procedures, macros, prototypes and models. The Action package defines a set of meta-model elements, whose goal is to represent the behavior descriptions at implementation level, e.g. statements, operators, conditions, characteristics.

The UI and KDM package was designed to represent the elements and the behavior of the GUIs (see figure 7 and figure 8).

UIRessources: It can be defined as Screen, report, UIField or UIEvent. Screens and Reports are the display units. UIField is a generic element to represent any field in a Screen or Report, as a field of text or drop-down list. UIEvents can be reported and associated with a uiAction.

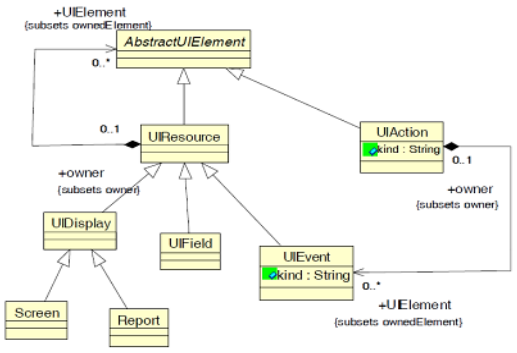


Figure 7: KDM metamodel. UI package (UIResources) (http://www.omg.org/spec/KDM/1.1/PDF/2009).

UIRelations defines the binding between the elements of a display and their content. There are two kinds of relationships: UIFlow and UILayout.

UILayout indicates the layout of a UIResource. It captures an association between two instances of display, one that defines the UI content, and the other that defines its layout.

UIFlow defines the behavior of the user interface as a sequential stream of a display instance to another.

The main objective of the UI package is to represent the logical structure of views, the spatial relationships between the elements of the UI (layout), and related events.

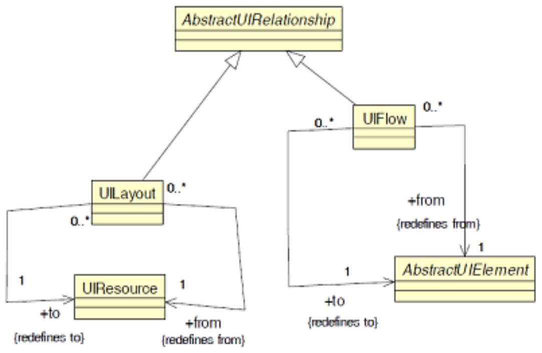


Figure 8: Figure 8: KDM metamodel. UI package (UIRelations) (http://www.omg.org/spec/KDM/1.1/PDF/2009).

IFML Model

The Interaction Flow Modeling Language (IFML) is an OMG specification (http://www.ifml.org/) for building visual models of user interactions and front-end behavior in software systems. The objective of IFML is the definition of Interaction Flow Models that describe the principal dimensions of the application view part.

An IFML diagram consists of one or more top-level view containers. Each view container can be internally structured in a hierarchy of sub-containers. A view container can contain view components, which denote the publication of content or interface elements for data entry (e.g., input forms).

A view component can have input and output parameters. A view container and a view component

Table 1: Task Operators.

Interleaving	The linked tasks can be performed concurrently,
Order independence	the tasks can be performed in any order
Synchronization	The tasks are concurrent and can exchange information among them
Parallelism	The tasks are performed in true parallelism
Choice	in this case it is possible to choose one task from a set of tasks
Disabling	the second task is deactivated once the first one has started
Suspend-Resume	The first task interrupts the second one. When it is finished, the second task can be reactivated from the state it was before the interruption
Enabling	when T1 completes it enables T2, when T2 completes it enables T3, and so forth through TN
Iteration	the task is performed iteratively: when it terminates, its execution is started again from the beginning
Optional	the task is optionally performed

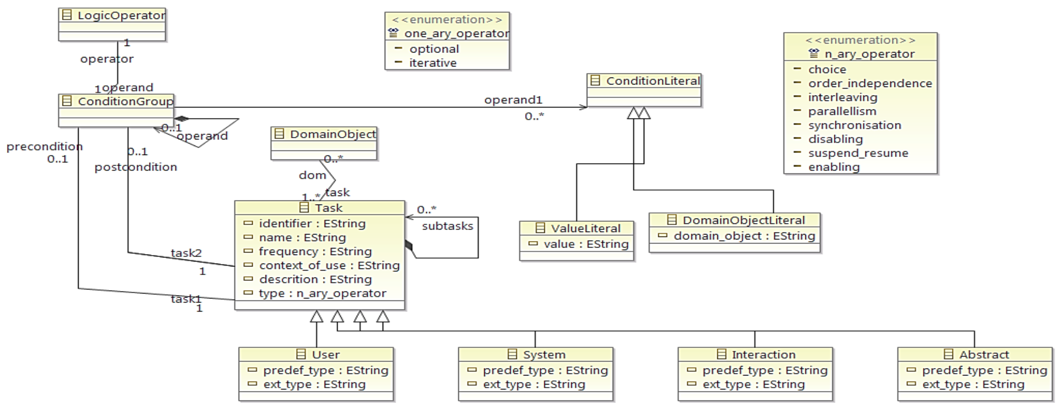


Figure 9: Task Meta-model.

can be associated with events, to denote that they support the user’s interaction. The effect of an event is represented by an interaction flow connection, which connects the event to the view container or the component affected by the event.

Task Model

The task of an existing system model is created with the aim of better understanding the design of the user interface. Its purpose is to describe how the activities should be carried out in an existing system, in order to understand its limits, problems and characteristics and so on.

A Task model consists of one or more top-level tasks (see figure 9), linked by operators in order to describe the relations between them. The operators include both N-ary operators and 1-ary operators. They are described in Table 1.

Tasks have many Categories:

- user task : an internal cognitive activity, such as selecting a strategy to solve a problem
- system task : performed by the application itself, such as generating the results of a query
- interaction task : user actions that may result in immediate system feedback, such as editing a diagram
- Abstract task: a task that has subtasks belonging to different categories.

3 CASE STUDY

3.1 Case Study Examples

We empirically evaluate the performance of our appr oach on three Java applications (see figure 10) with complex graphical interfaces to verify its applic ability.

3.1.1 First Case Study: Library System

It is a desktop management application. It can manage a database library with user interface. An advanced library system which contains best feature with multi-threaded operation without fail. This application is multi-threaded. It means that user can do many task simultaneously. The system allows users to perform the usual actions of adding and listing books/members details. Furthermore, it allows users to search for a book/member by its detail.

3.1.2 Second Case Study: Student Registration

It is a Desktop Management Application. It manages the student’s registration. To register a new student, he is invited to enter his name, password, date of birth, mobile number, e-mail, region, nationality and choose the gender and semester.

3.1.3 Third Case Study: Student Management System

The Student Management System can handle all the details about a student. The details include college details, course details, students personal details, academic details etc. The Student Management System is an automated version of manual Student Management System.

Table 2 shows the characteristics of our test syste m. For this, we use a Java software testing tool; CodeProAnalytix. (https://developers.google.com/ja va-dev-tools/codepro/doc)

The table also indicates the files size to give an

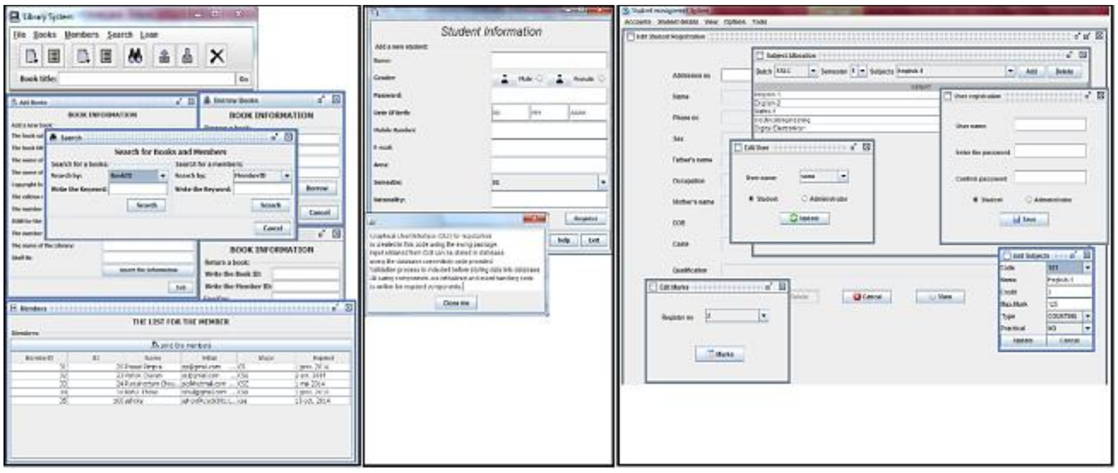


Figure 10: Case Study interfaces.

idea on the amount of analyzed data, as well as the number of widgets and treated events, which helps us to control and ensure the quality of our approach.

3.2 Process Result

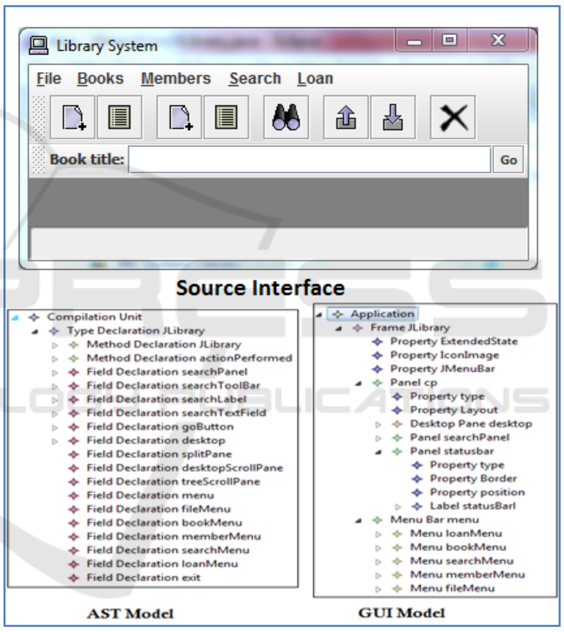
Our goal is to develop an approach based on the ADM for the modernization of existing graphical user interfaces. Our approach is described in Figure 2. Generally it is able to automatically generate graphical user interfaces abstract models.

Figure 11 shows the result of the model discovery phase presented by ASTM and GUIM models that was applied to the main window of the Library System's application. ASTM and GUIM models define the GUI components knowledge, their interrelations and the structural aspect of the application. They also encapsulate the events attached to each graphical component.

As depicted in figure 11, the model on the left is about the class structure for the selected graphical interface. The model on the right focuses on the presentation layer. It describes containers and widgets of the main frame and their properties. Figures below provide the result of the restructuring phase that represents the abstract level of the migration process of the migration process.

Figure 12 defines a model to model transformation result, that generate a platform independent model KDM, which is the most appropriate solution to represent and manage knowledge involved in a modernization process.

This KDM model represents the logical structure of the view and the spatial relationships between the user interface component represented by the UILayout element that defines a link between a resource and its layout. For example the UILayout


Library System
File
Books
Members
Search
Loan
Book title:
Go
Source Interface
Compilation Unit
Type Declaration JLibrary
Method Declaration JLibrary
Method Declaration actionPerformed
Field Declaration searchPanel
Field Declaration searchToolBar
Field Declaration searchLabel
Field Declaration searchTextField
Field Declaration goButton
Field Declaration desktop
Field Declaration splitPane
Field Declaration desktopScrollPane
Field Declaration treeScrollPane
Field Declaration menu
Field Declaration fileMenu
Field Declaration bookMenu
Field Declaration memberMenu
Field Declaration searchMenu
Field Declaration loanMenu
Field Declaration exit
Application
Frame JLibrary
Property ExtendedState
Property IconImage
Property JMenuBar
Panel cp
Property type
Property Layout
Desktop Pane desktop
Panel searchPanel
Panel statusbar
Property type
Property Border
Property position
Label statusBarl
Menu Bar menu
Menu loanMenu
Menu bookMenu
Menu searchMenu
Menu memberMenu
Menu fileMenu
AST Model
GUI Model
Figure 11: Example of code parsing result.

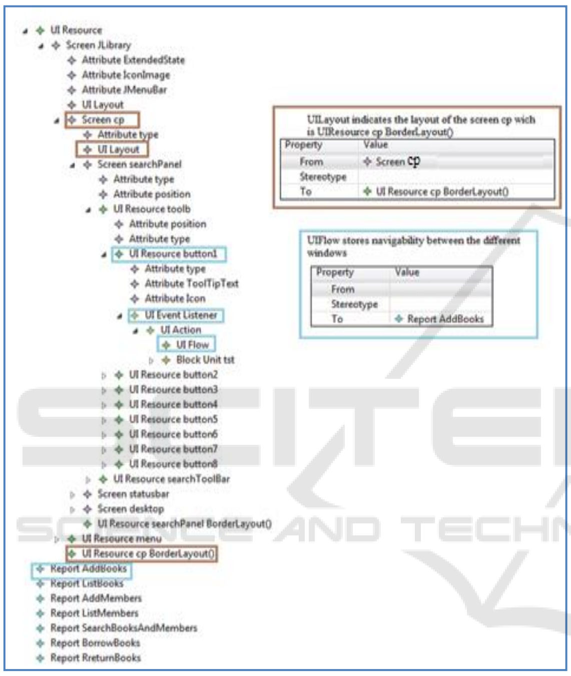
element in figure 12 links the Screen cp resource with its layout which is stored in the UIResource cpBorderLayout. There is also the UIFlow element that describes a behavior of the system, as figured in figure 12. After the click event on Button1, a navigation action is generated, presented by the UIFlow element, which has as property 'To' that contains the window to open represented by the AddBooks Report.

Figure 13 shows the model to model transformation result represented by the IFML model which specifies only the interactions between user interfaces.

Table 2: Test System.

System	Language	Loc (Line of code)	size	Windows	Widgets	Events
Library management	Java Swing	3324	11 Mo	10	280	31
Registration	Java Swing	1529	10 Mo	3	50	16
Student management	Java Swing	3268	9 Mo	15	370	50

NavigationFlow, or navigation manager, is generated when events are triggered. The NavigationFlows connect events with Interaction Flow Elements; the navigation target.



UILayout indicates the layout of the screen cp wich is UIResource cp BorderLayout()

Property	Value
From	Screen cp
Stereotype	
To	UI Resource cp BorderLayout()

UIFlow stores navigability between the different windows

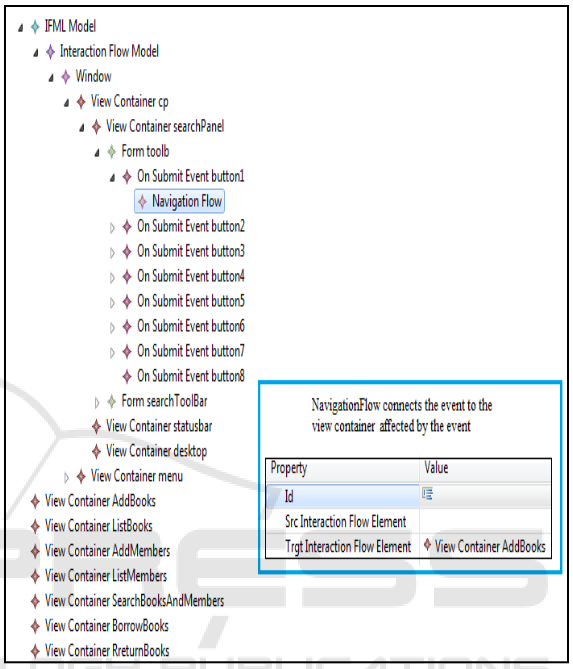
Property	Value
From	
Stereotype	
To	Report AddBooks

Figure 12: Restructuring KDM Result.

In Figure 13 the NavigationFlow element represents a connection that links the event 'OnSubmitEvent button1' with the 'ViewContainer AddBooks' affected i.e. the view container that will be displayed after the execution of this event.

Figure 14 presents the task model generated from a model to model transformation.

It contains the main possible tasks, their relationships and how to perform activities to reach users' goals. In this figure we see the way how these tasks were carried out. For example the task "Insert the information" will be executed only after the end of the execution of the previous task which is "Edit Information".



NavigationFlow connects the event to the view container affected by the event

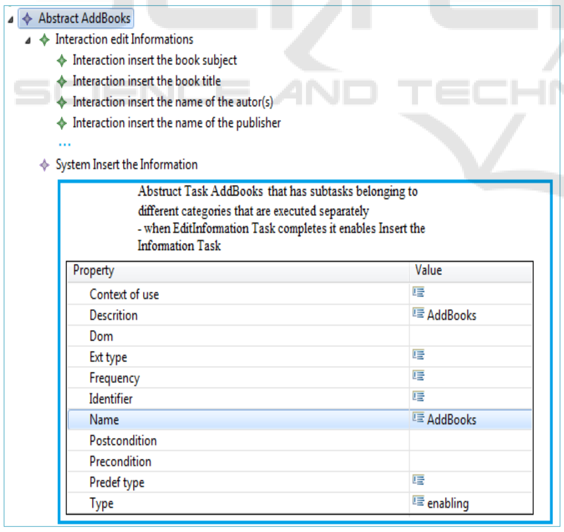
Property	Value
Id	
Src Interaction Flow Element	
Trgt Interaction Flow Element	View Container AddBooks

Figure 13: restructuring IFML result.

We can see that our approach can be applied to any program based on the same GUI library. We note that the results were well produced, which means that our analysis is applicable to major programs. The results of our empirical studies show that our approach is effective.

4 ANALYSIS AND DISCUSSION

KDM is the key element to the ADM approach; it is a complex meta-model that is used to model all of the software system artifacts. It allows the structural and semantic aspect representation of an application in a high level of abstraction. It represents and manages the knowledge involved in a modernization process, knowledge related to user interfaces of the existing software system. It also defines a specific model used to statically represent the main components of the user interface of an existing system. However IFML aims to express the behavior of content, interaction and user control of the front-end applications. It provides a description of the user interface regardless of the platform, focusing on the user interactions. It can express the events occur in the GUI. It can be considered as a better approach for modeling the behavior of the user interface independently of technology. But it does not describe the graphic elements and the relationships between them. While the task model determines how and when a task is executed, and how these results may impact on other parts of the system, it does not provide any details required for a complete description of the user interface. This analysis led us to have a good abstract representation of existing systems containing all necessary information needed during the migration process. This presentation is based on abstract combined models. KDM can represent and manage knowledge related to graphical user interfaces, while IFML is responsible to express the behavior of graphical user interfaces. The Modern interfaces can be induced from these two abstract models, while the third model TASKM will be exploited in the documentation of the generated interfaces. We are convinced that the combination of KDM, IMFL and TASKM will bring a good understanding and interesting evolution of existing software.



Abstract AddBooks
Interaction edit Informations
Interaction insert the book subject
Interaction insert the book title
Interaction insert the name of the autor(s)
Interaction insert the name of the publisher
...
System Insert the Information

Abstruct Task AddBooks that has subtasks belonging to different categories that are executed separately
- when EditInformation Task completes it enables Insert the Information Task

Property	Value
Context of use	
Descrition	AddBooks
Dom	
Ext type	
Frequency	
Identifier	
Name	AddBooks
Postcondition	
Precondition	
Predef type	
Type	enabling

Figure 14: restructuring Task model result.

5 RELATED WORK

Software modernization is a specific kind of evolutionary maintenance paradigm to solve reengineering problems. Much research both on model driven engineering and software modernization has been conducted. In (Mbarki et al., 2015) the authors develop a tool based on ADM approach allowing automatic reverse engineering of Swing GUI to obtain a RIA GUI. This tool allows the extraction of knowledge about GUI elements that can be represented in a KDM Platform Independent Model.

In (Rodriguez-Echeverria et al., 2014) the authors define a model-driven reengineering process of Legacy Web Applications. In order to abstract the extracted information and organize it according to the structure of a platform-independent metamodel, they used IFML.

To obtain a good understanding of system user interfaces, an example of reverse engineering techniques was described in (Stroulia et al., 1999). The authors proposed a process to obtain a user tasks model based on screen features analysis and on the tracing of user interactions with the system.

The modernization process is based on the analysis of the legacy application code. Two forms of this analysis were introduced: static and dynamic analysis. Concerning the static analysis, in (Silva et al., 2007) and (Staiger, 2007) the authors describe the static analysis of GUI code for reverse engineering purposes, with a focus on a detection of GUI elements and their relationships, exemplified in the swing, GTK and Qt frameworks.

In (Silva et al., 2010) they have also applied the static analysis in order to extract the behavior model of spreadsheet systems. They used a reverse engineering tool, named GUISurfer to infer models of interaction. Static Reverse Engineering analysis on the source code was performed also by using MoDisco (http://www.eclipse.org/MoDisco). It represents an extensible framework to extract information from an existing system. Using its plug in eclipse, the KDM and UML Model can be generated from the source code.

In dynamic analysis side, a dynamic process named GUI Ripping has defined. It dynamically builds a running GUI model to facilitate test case creation (Memon, 2003). It extracts sets of widgets, properties and value. The dynamic Reverse Engineering analysis was performed also by the Diver tool. Diver is a dynamic analysis tool for Java; it visualizes program's runtime functionality using sequence diagrams. (http://marketplace.eclipse.org/content/diver-dynamic-interactiveviews-reverse-engineering).

6 CONCLUSIONS

One of the major challenges in the migration of the legacy system artifacts process is the definition of an approach that allows a complete capture of various aspects about tasks, presentation and dialog structures and behaviors of the design knowledge, needed for the construction of the future user interface (UI). For that we used a static and dynamic analysis to obtain knowledge of the structure and behavior of source code. Our based ADM approach gives a solution that generates three independent platform combined models for good understanding and evolving the existing software assets. The process provides mechanisms and transformations in several steps, for analyzing the structure and behavior of the system objects. The resulting models contain all necessary information presented in a higher level of abstraction. This work should be extended to complete the migration process toward a modern specific platform.

REFERENCES

CodePro Analytix, https://developers.google.com/java-dev-tools/codepro/doc/

Eclipse, Diver, http://marketplace.eclipse.org/content/diver-dynamic-interactiveviews-reverse-engineering.

Harman, M., & Hierons, R., 2001. An overview of program slicing. *Software Focus*, *2*(3), 85-92.

JDT, Eclipse Java development tools, https://eclipse.org/jdt/

Mbarki, S.,Laaz, N., Gotti, S. & Gotti, Z., 2015. ADM-Based Migration from JAVA Swing to RIA Applications. In *ICIST 2015, The 5th International Conference on Information Systems and Technologies, Istanbul, Turkey.*

MBUI -Task Models, http://www.w3.org/TR/task-models/

Memon, A., Banerjee, I., & Nagarajan, A., 2003. GUI ripping: Reverse engineering of graphical user interfaces for testing. In *null* (p. 260). IEEE.

Eclipse, MoDisco, http://www.eclipse.org/MoDisco/

OMG, Abstract Syntax Tree Metamodel, http://www.omg.org/spec/ASTM/

OMG, Architecture-Driven Modernization, http://adm.omg.org.

OMG, Architecture-Driven Modernization: Knowledge Discovery Meta-Model, v1.1, http://www.omg.org/spec/KDM/1.1/PDF/2009.

OMG, Interaction Flow Modeling Language, http://www.ifml.org/

Rodriguez-Echeverria, R., Pavón, V. M., Macías, F., Conejero, J. M., Clemente, P. J., & Sánchez-Figueroa, F, 2014. IFML-based Model-Driven Front-End Modernization.

Sánchez Ramón, Ó., Sánchez Cuadrado, J., & García Molina, J., 2010. Model-driven reverse engineering of legacy graphical user interfaces. In *Proceedings of the IEEE/ACM international conference on Automated software engineering (pp. 147-150).* ACM.

Silva, J. C., Campos, J. C., & Saraiva, J., 2006. Models for the reverse engineering of java/swing applications. In *ATEM 2006, 3rd International Workshop on Metamodels, Schemas, Grammars and Ontologies for Reverse Engineering, Genova, Italy*.

Silva, J. C., Campos, J. C., & Saraiva, J., 2007. Combining formal methods and functional strategies regarding the reverse engineering of interactive applications. In *Interactive Systems. Design, Specification, and Verification* (pp. 137-150). Springer Berlin Heidelberg.

Silva, J. C., Silva, C. E., Campos, J. C., & Saraiva, J. A., 2010. GUI behavior from source code analysis. In *Interacç ao 2010, Quarta Conferência Nacional em Interacçao Humano-Computador, Universidade de Aveiro*.

Staiger, S., 2007. Reverse engineering of graphical user interfaces using static analyses. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on* (pp. 189-198). IEEE.

Stroulia, E., El-Ramly, M., Kong, L., Sorenson, P., & Matichuk, B., 1999. Reverse engineering legacy interfaces: An interaction-driven approach. In *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on* (pp. 292-302). IEEE.