KDM- As a tool for modernizing Legacy Systems

Amit R. Wasukar

Abstract— The performance of system depends on how it is designed and for what functionality it is developed, but the fact lies in architecture of the system. Because legacy systems have a complex type of architecture compared to the new age systems. So many a times handling legacy systems requires more care, otherwise it can lead to improper functioning. To deal with this situation we can use the concept "Architecture Driven Modernization (ADM)", the reason for handling this situation is due to the archival information that resides in legacy systems. ADM is used for modernizing the legacy systems by analyzing its artifacts and through this we can transform the architecture of the existing system to the new one as per user or organization needs. To make it possible, we can use a specification provided by ADM called as "knowledge discovery metamodel (KDM)". Knowledge Discovery Metamodel supports many programming languages because it is an intermediate representation of software artifacts and its operational environment. In this paper we are trying simplify the intermediate representation of legacy system using KDM with its respective models.

Index Terms — ADM, Artifacts, Architecture, Knowledge, KDM, Legacy systems, Modernization, Architecture Knowledge.

1 Introduction

The need for transforming old systems into the new systems is increasing rapidly because of the inventions that are carried out in this field. Inventions in the sense that faster systems are build up using latest technology which wipes out the old technology. But in many cases i.e. in many organizations, the use of old systems and their respective work played a vital role, so we cannot remove these systems directly from the working chamber of organization. The reason for this is the data and information they possess for the proper functioning of any organization, we can call this information as "archival type of data". Legacy systems and old systems both can be accrediated same and can be used interchangeably. The functioning, characteristics and working of legacy systems are totally different than the ones which are present today; still they are useful because of the key capabilities they possess. Legacy system can be a application program or a computer system which are designed to work for some specific job, also in many situations it is treated as the technology or method used for calculating/evaluating some function. The legacy system may or may not remain in use, even if it is no longer in use, it may continue to impact the organization's functionality due to its historical role data. So there is a need of some mechanism which helps in converting or modifying the legacy systems into the new system. As we all know systems which are building over time are dynamic in nature, because of the technology they used and for what purpose they are developed. Technology is the changing wheel of time since after every random time span it is upgraded or changed, this also is the main reason for transforming the legacy sytems. But one can think that why it is given so much importance and why not replace old system directly with new system, reason for this is cost and expensiveness attached to it. Another reason is complex architecture of legacy system and insufficient manpower to understand it i.e., first we have to train the programmers to understand legacy system architecture and then again build another system with same functionality. So to avoid this we can use the modernization technique which helps in minimizing the cost related with it.

Architecture is one of the important thing related to the systems either it is old or new. Complex architecture often leads to the ambiguous development of system. In different terminology architecture is defined differently, like theoretical point, organizational point, software & hardware point, etc. In this paper we are trying to evaluate the solution for transforming old system architecture into new one. To achieve this there are many techniques available from which some are old, but the efficiency of technique is not determined by just observing that it is old or new, it is judge by the performance and efficiency provided by that technique.

One thing to note here that architecture related decisions are the first to be taken into consideration during system development and ultimately they virtually affect later stages of the system development process, also the impact of architectural mistakes results in high economical risk. So to deal with solution Object Management Group (OMG) invented a process called as "Architecture Driven Modernization"[3][6], which is used for understanding and evolving the existing software artifacts or assets[4]. We will explain what is ADM in further sections.

Everyday a new technology/technique is developed which satisfies the needs of users as well as organizations there is lack of interdependency between the uses of these technologies. In fact in many cases it happens that organization is ready with updated version of their software with some improved functions which helps user in many perspectives, and to use this, user needs to make some exchange operations with their existing systems, but a section of users does not like this i.e., it isn't attract them. Reason for this is the extended workload imposing on them, so they try to ignore it which ultimately degrades their system performance as compard to other competitive organizations software. Form this scenario we understand that to make users happy we have to come up with new ideas with some simple trics to modernize their existing systems. While modernizing the system, main thing we need to take into account that it does not affect the basic functionality of system.

So to deal with all this, we can use "Knowledge Discovery

• *Amit R. Wasukar, PG student, Dept. of Computer Engineering, Dr. B.A.T.University, Lonere, India, PH-+91-9766881927.*
E-mail: amitwasukar@gmail.com

MetaModel (KDM)" as an intermediate representation. Today, many researchers are focusing on this transformation using different specifications and standards, but the first one which is standardized by ISO is KDM[1][2]. KDM is a standard invented by OMG under ADM process and mainly used because it has the capability to seperate between different physical artifacts related to the system. UML plays an important part in the modernization process as it is used for modeling purpose. Modelling is the designing of software applications before coding.

In further sections we will elaborate the details regarding the modernization of existing systems i.e., legacy systems. In section 3 we see what is mean by architecture of software system and introduction to Legacy systems. In section 5 we try to explain what is KDM and its representation, in section 6 modernization process is explained and finally we conlude the paper with future work.

2 Software Architecture and Legacy Systems

In todays world the need for making the systems more agile and exible is increasing rapidly with the intent of interoperability, language independant and platform indepenant. There are existing systems which plays a vital role in organizations growth though they are old. The functioning, characteristics, working and complexity of these systems are totally different than the ones which are present today; still they are useful because of the capabilities they possess. One of the reasons for their dissimilarity is the"Architecture" they use. These systems can be called as "Legacy Systems". The legacy system may or may not remain in use even if it is no longer used, it may continue to impact the organization due to its historical role. The data that resides and processed through Legacy systems is archival type of data that can not be directly used and processed in the modern systems. A legacy system may include procedures or terminology which are no longer relevant in the current context, and may hinder or confuse understanding of the methods or technologies used.

Software architecture is helpful in deriving the important and useful characteristics of software system along with its functionality. When we define any architecture we must take into account that the requirements should be fullfiled and there is a scope for modernizing that architecture. In many cases the conflict arises because of amibiguity of required architecture due to which designers and programmers fell short to implement the needed software. In terms of IT it is a set of structures required to explain about the software system, which comprise software elements, the relations between them, and the properties of both elements and relations. OR it is the conceptual model that defines the structure, behaviour, amd views of systems [1][5]. In general, we can term architecture as "the process and product of planning, designing and construction". In software development, planning, designing and construction plays very vital role sice these are the basic operations involved in software development. In a broad way architecture of a system is termed as

- The high level structure of a software system
- The way of creating such a high level structure
- Making documentation of such a high level structure

Here we explain what legacy system is and what software architecture is. The reason behind that software architecture impacts overall working of the legacy systems and in further sections we get a clear idea why software architecture and and why legacy system?

3 Architecture Driven Modernization- ADM

The necessity for modernizing and transforming old systems is becoming one of the important factors in the organizations. To deal with this problem Object Management Group (OMG) has discovered a new concept called as "Architecture Driven Modernization" (ADM)[6]. ADM is a technique which is helpful in making systems architecture more agile also helped in making platform and language independent systems. Through ADM we can get a number of advantageous services/applications like

- Software Improvement
- Interoperability
- Reuse
- Modifications
- Migration
- Translation
- Integration
- Service Oriented Architecture

That is, through these services system maintenance becomes some what easy. One thing to take into consideration is the inability of existing systems to cop-up with the changing technological environment, this many a times reflected into the progress of organization i.e., it slows down the functioning of the company. So above activities which collectively define the basis of ADM can easily overcome the existing systems problems[3].

In large context ADM is the process of understanding and working out with existing software assets. ADM is the best path for moving an Aging System to an Agile System when used with its best scenarios and standards. ADM is used when existing IT practices fail to deliver against business objectives. Business objectives include the competition among organizations, survival in the market.

ADM is one type of task force which is used by many vendor and user organizations for sharing the tools and analyzing the meta-data from existing systems environment. ADM defines a standard which isused for the purpose of sharing tools and artifacts related to existing systems named **"Knowledge Discovery Meta-Model (KDM)".** Mainly ADM helps us to make systems

- Platform Independent
- Language Independent
- Interoperable

These are the important factors which enables organization to widely use ADM because today all we need is the faster executable software systems and we can get it if it satisfies the above criteria i.e., the system should be platform and language independant. At any time it is interoperable between different situations.

4 Knowledge Discovery Metamodel - KDM

As name suggests Knowledge Discovery Metamodel is related to the knowledge representation of the existing system through some intermediate representation format. In many cases i.e., in various organizations it is very important to make an old system reusable, so that cost for developing new systems gets reduce. This is because old systems/existing systems holdlots of archival type of data/information which plays a vital role in organizations growth.

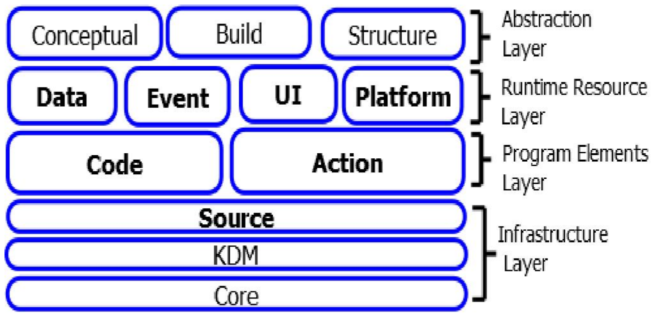
Knowledge Discovery Meta-Model provides a common interchange format for the interoperability between existing software systems. KDM is a metamodel which helps us to represent information associated with existing software system, its elements, associations and operational environment. Because of the common metamodel that KDM provides, each vendor tool can be able to exchange common views across platforms and languages for the purpose of analyzing, standardizing and transforming existing systems[1].

Main reasons for selecting KDM as transforming medium are vary from application to application because of the diversity in its development. KDM is standardized by the ISO as one of the standards provided by ADM. KDM has the capability to handle different types of language constructs. Like through KDM we can transform COBOL, C, and PASCAL language legacy code into the modern language like JAVA. Along with this it is also possible to modernize C and C++ into Java code and vice-versa. In process of transformation, first we have to convert legacy source code into the KDM representation which is a common interchange format and by using this we can transform one language legacy system into the other language system.

Taking all these points into consideration if it is possible to modernize/transform the existing system, we can reduce a considerable amount of work, cost, manpower, programmers overhead. Fortunately today we are having some of the better utilizing equipments, procedures, techniques, tools, methods, and representations for making this happen as compared to the old age where not much resources and representations are available.

4.1 Layered Architecture of KDM

KDM is represented and understand defined by the layered architecture which contains different abstraction layers for different purposes. Each layer is organized and divided by its



Layers & Packages In KDM

functionality and some packages[1].

Fig. 1 Layered Architecture of KDM

The KDM architecture comprises four layers as shown in fig 1;

- Infrastructure Layer
- Program Elements Layer
- Runtime Resource Layer
- Abstraction Layer

4.1.1 Infrastructure Layer

This layer is at the lowest abstraction level, which consists of a small set of common metamodel elements (such as entity and relationship) used through entire KDM levels. This layer consists of three packages: *Core, KDM* and *Source.*

4.1.2 Program Elements Layer

It represents the code elements and their associations. It consists of a broad set of metamodel elements common between different programming languages to provide a language independent representation. There are two packages in this layer: *Code* and *Action.*

4.1.3 Runtime Resource Layer

It represents higher level knowledge (such as operational environment) about existing software systems. This kind of knowledge cannot be extracted from the syntax at code level but rather from the runtime incremental analysis of the system. There are four packages in this layer: *Data, Event, UI* and *Platform.*

4.1.4 Abstraction Layer

It defines a set of metamodel elements for representing domain and business specific overview of the system. Extracting this kind of knowledge involves input from experts and analysts. *Conceptual, Structure* and *Build* are the three packages present in this layer.

Each layer has its own functioning and working parameters related to the software architecture modernization. The key design characteristics of KDM are[1][4]:

1. KDM specifies ontological structure for describing existing software systems assets.
2. KDM identifies entities in the source code and actually work as an Entity-Relationship Model.
3. KDM has the capability to capture language specific, application specific and implementation specific entities and relationships.
4. KDM models are composable since it is possible to group many entities into one container to represent one single entity.

4.2 Packages in KDM

KDM contains 12 packages, each package is designed by one or more class diagrams and defines a set of metamodel elements whose purpose is to represent a certain independent facet of knowledge related to existing software systems. These packages are shown in fig.1, we can enlist and categories them with respect to their layers. From these packages not all are important; the packages which are useful are Source, Code, Action, Data, Event, UI and Platform. The reason behind this is functioning they can perform[1]. We explain only these packages and analyse them next sections.

4.2.1 Source Package

This model identifies and recites the physical artifacts of a legacy system (like source files, images, configuration files, resource files, and so on) as KDM entities and defines a mechanism to link those KDM entities and their original representation in the legacy source code, for which the KDM representation was created.
Defines *Inventory Model Of KDM Source Package.*

4.2.2 Code Package

The code package defines a set of CodeItem elements that represents the common named elements in the source code supported by different programming languages such as data types, classes, procedures, methods, templates and interfaces.
Defines *CodeModel Of KDM Code Package.*

4.2.3 Action Package

The action package extends the code model by means of more metamodel elements that represent behavior descriptions, control and dataflow relationships between code elements. The action package adds two key elements: the ActionElement and the AbstractActionRelationship. The ActionElement metamodel element depicts a basic unit of behavior and represents some programming language constructs such as statements, operators and conditions. The AbstractActionRelationship usually represents the use of a name in a statement.
Defines *Code Model Of KDM Action Package.*

4.2.4 Data Package

The data package defines the representation of several data organization capabilities. This representation is related to the persistent data aspects of a legacy system. Therefore, this package makes it possible to represent complex data repositories like relational databases, record files and XML schemas and documents.

4.2.5 Event Package

The event package defines a set of metamodel elements for representing state and state transitions caused by events. This package represents two kinds of states:

- concrete states that are explicitly supported by specific state machine based languages and
- Abstract states related to a specific algorithm, resource or user interface.

4.2.6 UI -User Interface Package

The UI package provides metamodel elements for representing the resources related to user interface aspects, such as their composition, their sequence of operations and their relationships to the legacy systems.

4.2.6 Platform Package

The platform package defines artifacts related to the runtime platform and environment of a legacy system such as inter process communication, the use of registries, the management of data, and so on. The platform metamodel elements depict the execution context of the legacy code, since a legacy system is not only determined by the programming language of the source code, but also by the selected runtime platform.

5 KDM Transformation And Representation for Legacy Source Code

KDM's purpose is to define and illustrate a standard for representing software systems and information systems like representing software artifacts which includes source code, user interface, database, image files, configuration files, binary files, etc. KDM works in a bottom up manner i.e., first we analyse the source code of legacy system and from that we create a KDM representation; we can show it by explaining following example. In this diagram we define simple language code having a class, named Paper with data member **'test'** which is integer data type and a member function '**avg**' with float data type.

```
class Paper          {
            int test;
            float avg()    {…… }
```

Fig. 2 Source Code Example (Class example)

Now, here comes the main idea of paper, suppose we have legacy source code of thousands LOC written in any language and it is very much needful for the organization to use it properly but in many cases it happened that new age programmers and analysts are incapable to understand this code or they may required much more to grasp this, so to minimize this task we can use KDM. KDM is used as an intermediate representation of the legacy software system with common interchange format. Figure 3 shows KDM representation derives from UML as UML is a meta-modeling language.

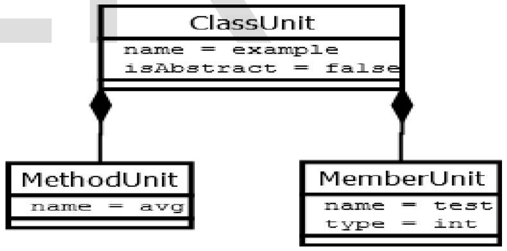


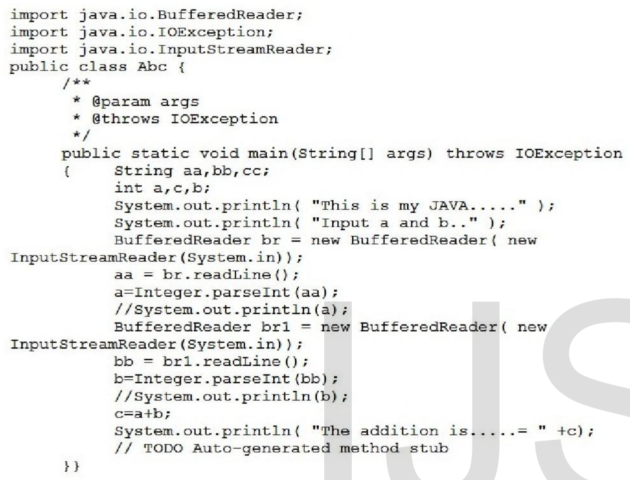
Fig. 3 KDM representation separating its artifacts

From above diagram we get the idea that KDM is capable for evaluating different physical artifacts related to the software system. If we look at our example then some of the artifacts extracted are classes, methods, datatypes present in the code, i.e., we can clearly understand thewhat is class name, what are its different member functions with its datatypes, which are data members used to store values. So programmers can directly analyze the source code by looking into the KDM representation and easily modernize it according to specified requirements. By this, many things can reduce i.e., cost for analyzation, headache for programmers, time for organization to sustain in the market. Therefore it is easy to transform the legacy code into the new language code by simply analyzing its artifacts related to the specific target language.

5.1 Basics for Transformation of Legacy Source Code Using KDM

Our concept is to modernize the legacy source code into the new one using KDM representation and to achieve that many viewpoints should be taken into consideration.
Goal - Legacy systems should be transformed/modernize into the Target system.
Method – Use KDM as a common interchange format for extracting the physical artifacts of legacy software system.
For modernization purpose we take a simple source code written in java and represent it in KDM format so that we can analyse its artifacts.

5.1.1 Legacy Source Code

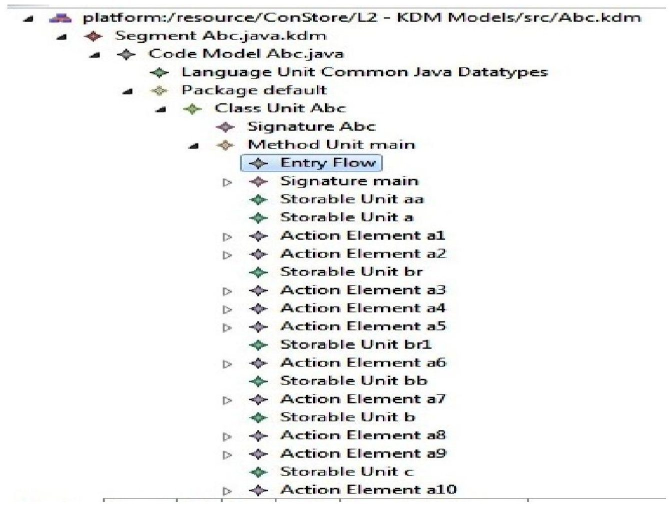


```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class Abc {
      /**
       * @param args
       * @throws IOException
       */
      public static void main(String[] args) throws IOException
      {     String aa,bb,cc;
            int a,c,b;
            System.out.println( "This is my JAVA....." );
            System.out.println( "Input a and b.." );
            BufferedReader br = new BufferedReader( new
InputStreamReader(System.in));
            aa = br.readLine();
            a=Integer.parseInt(aa);
            //System.out.println(a);
            BufferedReader br1 = new BufferedReader( new
InputStreamReader(System.in));
            bb = br1.readLine();
            b=Integer.parseInt(bb);
            //System.out.println(b);
            c=a+b;
            System.out.println( "The addition is.....= " +c);
            // TODO Auto-generated method stub
      }}
```

Fig. 4 Legacy Source Code

5.1.2 KDM Representation

From above code we can get KDM representation as shown in figure 5. It depicts the basic information regarding the source code.



```
platform:/resource/ConStore/L2 - KDM Models/src/Abc.kdm
  Segment Abc.java.kdm
    Code Model Abc.java
      Language Unit Common Java Datatypes
      Package default
        Class Unit Abc
          Signature Abc
          Method Unit main
            Entry Flow
            Signature main
            Storable Unit aa
            Storable Unit a
            Action Element a1
            Action Element a2
            Storable Unit br
            Action Element a3
            Action Element a4
            Action Element a5
            Storable Unit br1
            Action Element a6
            Storable Unit bb
            Action Element a7
            Storable Unit b
            Action Element a8
            Action Element a9
            Storable Unit c
            Action Element a10
```

fig. 5 KDM epresentation

As there is no image files or binary files present so no need of inventory model. Code model gives us details about datatypes, member functions, packages, etc present in the code. And through this we can easily modernize program code into another language code with its proper functioning. Here we are not explaining all the working in detail because it is out of scope of this paper.

Therefore by we can modernize the legacy system to new system KDM representation and it is a concept for doing this, still this concept needs some more acceptance and observations. In next section we just introduce the concept of modernization process used by KDM for transformation purpose.

6 Modernization Process

Normally modernization starts where existing organizations fails to do deliver against its requirements. The factors that fall short include new application development, application integration and maintenance of applications. Modernization helps, essay and discovers the refactoring, redesign and redeployment of vital application architectures for meeting critical business requirements to lowers risks, costs and delivery timeframes. Modernization backs many standard scenarios like:

- Application quality improvement
- Portfolio management
- Source to source coversion
- Platform migration
- Application and System integration
- Service oriented architecture
- Model driven architecture

When we convert any old thing into the new one we usually come across many problems in which some are not avoidable, to deal with this situation a solution must be available. If it is possible that existing systems or legacy systems can be converted into the modern one with the help of some tool or mechanism in a simple way then the cost and time of the users and organizations can get reduced. In this way we are able to change state of the old system into the new one, this concept can be called as the "Modernization".
Here we trying to modernize the legacy systems with the help of KDM which is a standard specification defined under ADM which is introduced by Object Management Group (OMG). The process for modernization can be understood by following diagram.

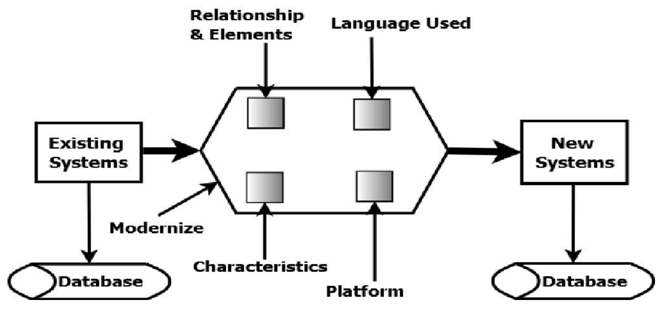


Fig. 6 Modernization Process for Legacy Systems

As shown in figure, we have two systems in hand first one is the "Legacy System/Existing System" which we have to modernize and second one is "New System" to which we have to modernize. We can understand from diagram that both systems use the database but each have its own version, compatibility and uses. If we want to transform/modernize the system then all the factors related to system should be taken into consideration, like which language is used, what is the relationship between entities and its elements, what platform is used by the system (either for old system or for modernize system) and what are the characteristics they possess to work or function properly. Now by observing the above diagram one thing is clear that modernization process needs to take into account the various factors associated with it.

- Relationship & Elements
- Characteristics
- Platform
- Language Used

By using this process it becomes easy to modernize the legacy system with some major/minor changes done in the KDM representation of both systems.

7 Conclusion

By making the legacy systems, old systems usable or workable with some minor, limited changes to its artifacts and architecture, then many things will get easy for organization's development and user also. To achieve this KDM can play a vital role because of its functionality. KDM is developed under ADM which is a technique used for modernization of the system. By using KDM we can analyse the architecture of legacy system and further we are able to extend it to the web based application system. The layer organization of the metamodel of the KDM standard enables knowledge representation of all the software artifacts (source code, databases, user interfaces, business rules, etc.). KDM helps us to extract the various artifacts related to the system from analyzing its source code and then can be represented in its own form which is one type of common interchange format. The knowledge recovered from legacy systems and represented in KDM models can be shared by different modernization tools according to the KDM ecosystem. KDM includes various models for various operations; all these models have their own functionality with respect to the code. By knowing all the artifacts from source code we can made changes to it easily and today market needs this type of transformation to make organization's competition sustain, alive and ultimately this will be benefited to the growth of organization and users also.

References

[1] Ricardo Prez-Castillo, Ignacio Garca-Rodrguez de Guzmn and Mario Piattini, "Knowl-edge Discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems", Alarcos Research Group, University of Castilla-La Mancha, Pdela Universi-dad, 413071, Ciudad Real, Spain

[2] Information technology - Architecture-Driven Modernization, OMG. Architecture-Driven Modernization (ADM) / Knowledge Discovery Meta-model (KDM) 1.2 Specification. on (ADM): Knowledge Discovery Meta-Model(KDM)

[3] William M. Ulrich, "Architecture Driven Modernization 101: Concepts, Strategies & Justification", Tactical Strategy Group, Inc., www.systemtransformation.com

[4] W. Ulrich.(2004), "A status on OMG Architecture-Driven Modernization task force," In Proceedings EDOC Workshop on Model-Driven Evolution of Legacy Sys-tems (MELS), Monterey, USA [IEEE Computer Society Digital Library].

[5] OMG document, "Architecture-Driven Modernization (ADM): Knowledge Discovery Meta-Model (KDM)", http://www.omg.org/spec/KDM/1.1

[6] ADM whitepaper, "Why do we need standards for the modernization of existing systems?", OMG ADM Task Force

[7] Ga¨etan Deltombe Netfective Technology Software 32, avenue L´eonard deVinci 33600 – Pessac, France g.deltombe@netfective.com Olivier LeGoaer, Franck Barbier University of Pau Avenue de l'universit´e,Pau, France folivier.legoaer, franck.barbierg@univ-pau.fr ," Bridging KDM and ASTM for Model-Driven Software Modernization"