USE OF MODEL DRIVEN ARCHITECTURE AND TRANSFORMATION IN SPL

Krishna Murthy, Sachin — *skrish35@hawk.iit.edu* — *A20354077*
Krishnamurthy, Karthik — *kkrish14@hawk.iit.edu* — *A20344464*
Subraya, Shashank — *ssubraya@hawk.iit.edu* — *A20290656*
Nayak, Ajith Aravind — *anayak@hawk.iit.edu* — *A20357090*

***Abstract*— In this paper we present the working functionality of Traditional Software development model in SPL and the resulting drawbacks of it. Also we will describe our approach for the need for a new architecture and the reason for choosing Model Driven Architecture over other architectures and present various advantages to address the drawbacks of the working functionality of Traditional Software Development model in Software Product line.**

Keywords*— Model Driven Architecture, Software Product line, Platform Independent Model, Platform Specific Model, Computation Independent Model.

I. INTRODUCTION

A software product line (SPL) is a set of software intensive systems and they satisfy the specific needs of a particular market segment by sharing a common, managed set of features that are developed from a common set of core assets in a prescribed way.

The main objective of software product line is at developing the productivity and quality of software development by enabling the management of common and variable features of a system family as it is a software intensive product which is developed for specific domains.

Product Line Architectures are designed using Scope/Commonality/Variability (SCV) analysis which captures the key features such as (1) scope, defines the background of PLA, (2) commonalities, the attributes that repeats across all the members of the product family are defined, (3) variabilities, contains attributes which are unique when compared to the different members of the product family. [1]

Motivation for the Need of Model Driven Architecture in SPL :

Earlier the Traditional Software Development model used in SPL is a conceptual model used to develop information systems with well-defined phases. As these models lie on one end of the planning emphasis these models are called plan driven. But there are several drawbacks that existed in different areas of development life cycle and management style. Also, the low level of abstraction at which Product line architectures are designed now a days using traditional development approach inspite of the evolution in programming languages (C,C++,JAVA) and runtime platforms (J2EE,Web Services) is a major concern. This created the need for a new architecture that could resolve the drawbacks to be used in SPL. Hence the Model Driven Architecture was used in SPL whose main objective was to find an optimal solution for this problem.

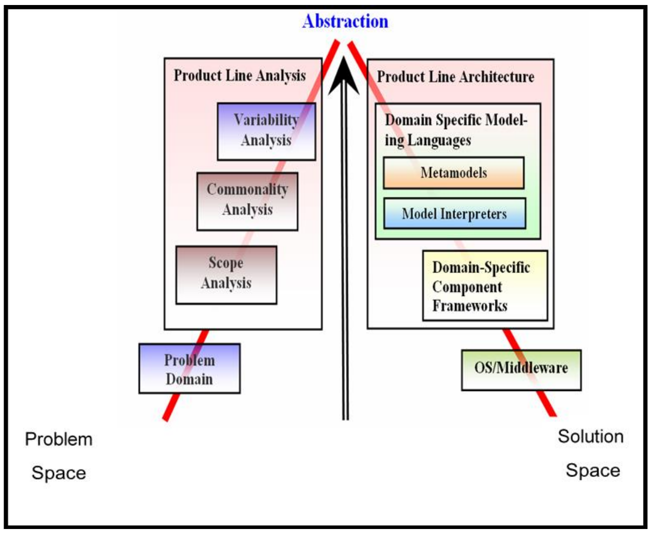

Fig 1: Using DSML and Domain specific component framework to enhance abstraction and narrow the gap between problem and solution space of software intensive systems.

Model Driven Architecture (MDA) is a software approach for the development of software systems. It provides a set of guidelines for the structuring of specifications, which are expressed as models. MDA is a framework in software product line which is defined by the object management group (OMG).

The term "architecture" in Model-driven architecture does not refer to the architecture of the system being modeled, but rather to the architecture of the various standards and model forms that serve as the technology basis for MDA.

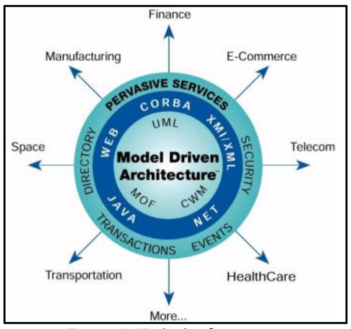

Fig 2: MDA Architecture

The above figure shows the Model Driven approach as the core architecture encompassing various technologies and services, and also the various fields such as manufacturing, finance, telecom etc. in which MDA has been incorporated.

A complete MDA specification consists of a definitive platform-independent base model, one or more platform-specific models (PSM) and sets of interface definitions, each describing how the base model is implemented on a different middleware platform.

The reason for choosing MDA over other architectures to be used in SPL is that it first focuses on the functionality and behavior of a distributed application or system, undistorted by idiosyncrasies of the technology platform on which it will be implemented. Therefore this will not create a necessity to repeat the process of defining an application or system's functionality and behavior each time a new technology comes along, whereas other architectures are generally tied to a new technology [2].

II. RELATED WORK

There has been a lot of research going on in areas of Software product line and software product line architectures [3]. They focus on automated assembly and customization of domain-specific components instead of manual reprogramming. Some of the early works on Model driven architecture includes [4] explaining the development and maintenance of a model driven product line architecture. [5] Provides a methodological approach to explicitly support variability in model-driven software development processes, model-driven support to the analysis phase of the development of variability intensive systems and to the architectural design phase of variability intensive systems using UML. In [6] a traceability meta model is used to link abstractions from the variability and the requirements models. This enables the navigation across abstractions of the different types of models using model-driven techniques and tools. The traceability model supports both backward and forward traceability between feature and requirement models.

Model driven Architecture came with its own challenges like capturing new requirements into existing MDD and migrating then existing architectures into MDD. [7] discussed a detailed solution for these challenges. Layer based PLA and toll supported model migration were introduced in this. [8] developed an evaluation framework for model driven product line engineering tools. It came up with 25 Evaluation characteristics and four tools namely Atlas Transformation Language (ATL), UML Model Transformation Tool (UMT, ArcStyler and XMF-Mosaic were evaluated. In 2010 [9] discussed variability modeling using delta models and formalizing delta modeling along with model refinement and configuration.

III. TRADITONAL SOFTWARE DEVELOPMENT APPROACH

The traditional software development approach has been used to develop projects since its inception. The two activities that a traditional software development process consists of are development and maintenance. These traditional development approaches assume that problems can be well defined, results can be optimized and results can be predicted. Traditional methods also known as the heavyweight methods perform extensive up front planning to control and measure the variations in the development cycle. They impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. [10]

Although the traditional approach has its advantages such as scalability and predictability, this model doesn't seem to fit in the world of evolving technologies.

Major Problems in Traditional Software:

The Productivity Problem:

Whether we use incremental and iterative version of this processor the traditional waterfall process, documents and diagrams are produced in the initial stages and these include UML diagrams like use cases, class diagrams, activity diagrams and so on. However most of the artifacts from these phases are just paper and nothing more.

So either we use our time in the first phases of software development building high level docs and diagrams, or we use out time in the maintenance phase finding out what the software is actually doing. Neither way are we directly productive.

The Portability problem:

In software industry, new technologies are being invented and are becoming popular each year, and sometimes even faster. Many companies need to follow new technologies. Tool vendors stop supporting old technologies and focus on the new ones and as a consequence of this the investment in previous technologies lose value and they may even become worthless. As a result the existing software is either ported to new technology, or to a newer version of an existing technology.

The Interoperability Problem:

Software systems rarely live in isolation. Most systems need to communicate often with already existing systems and it needs to get its information from existing back-end systems. Even when systems are completely built from scratch, they often span multiple technologies, sometimes both old and new. The different components are all built using the best technology for the job, but need to interact with each other. This has created a need for interoperability.

The Maintenance and Documentation Problem:

The maintenance of the traditional process is not very efficient as the maintenance activities have to be performed before the product is released in order to cope with the new technologies and changing customer requirements, by the developers. Documentation has always been a weak link in software development process. Writing documentation during development requires time and slows down the process. It does not support the developers' main task. This is one of the main reasons why documentation is often of not very good quality. With every change in the code, the documentation needs to be changed manually as well.

IV. MODEL DEIVEN ARCHITECTURE

The object management group has proposed a framework for MDE which is called MDA which aims to self-establish as one and only standard. MDA's main objective is to solve the changing integration and technology problem. Using models is the main idea behind it. This will make the system features and properties reflect and abstract description. Therefore, models won't be effected by the changes in technologies [11].

MDA also has a SDP (software development process), thus, its purpose is to develop software systems that are executable. Therefore, MDA is expecting to solve some problems along the way as well like low productivity and interoperability issues in software. This problem was handled by development and performing analysis of systems by using high end models which can automatically generate code. The interoperability issue on the other hand can also be handled easily as the techniques for code generation allows the programmers to obtain code for different technologies, and this leads to another advantage that is the models reusability.

Some most common concepts that MDA standard has defined are below: quoted from object management group:-

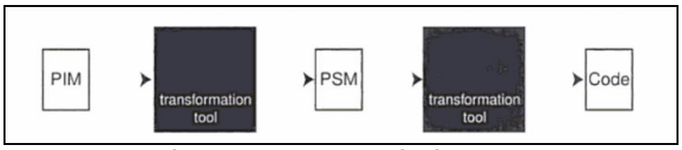

Fig 3: Three major steps in MDA development process

Platform: Platform is a set of technologies and subsets that provide a coherent set of functionalities via specified usage patterns and interfaces, which can be used by that platform without concerning about the functionality. [12]

Platform Model: A platform model provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided by that platform. It also provides, for use in a platform specific model, concepts representing the different kinds of elements to be used in specifying the use of the platform by an application.

Platform Independent Model: This model is a detailed view of the system which consists of more details from a Platform Independent Viewpoint. This model does not contain the Technical details or the platforms on which it may be implemented, but it contains details about the business functionality and its behavior.

Platform Specific Model: A platform specific model is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

Computation Independent Model: This model simply represents the system, without specific information about how it is to be implemented. This model represents what the system is supposed to do and the environment in which it will operate. According to the MDA specifications, CIM is usually a simple UML model which should be traceable to the PIM's and PSM's that implement them.

The system development according to MDA framework starts with the PIM construction. Later on, that PIM is transformed into PSM. Finally, PSM is used to generate the code.

The MDAs fundamental operation is to convert the PIMs in to PSMs. It is stated in the MDA guide that PSMs and PIMs models are expressed as UMLs transformations and models that must be automated as much as possible.

Automation of the Transformation Steps:

The MDA Process may look very similar to the Traditional development approach, but there are crucial differences. Usually, the transformation from model to model, or from model to code, are generally done by hand. There are certain tools that generate code from the model, but that usually is a template code and it still requires work to be done manually by hand. On the other hand, the MDA transformations are executed by tools and are able to transform a PSM into code.
The most important aspect of the MDA is that the transformation from PIM to PSM is automated. This is one of the main benefits of MDA.

Initially when writing the code, the MDA approach is very new. The Tools are not sophisticated enough to transform PIM to PSM and PSM to code with a hundred percent efficiency. But however the tools provide the basic functionality of creating and changing objects in the system and also generate a running application from a PIM. This allows the developer to have an immediate feedback on the PIM which is currently under development. This makes the Transformation steps easier and supports flexibility and saves a lot of time. Using the Transformation steps, the basic prototype of the system can easily be generated on the fly.

There are two types of transformation horizontal and vertical. If the source model belongs to the same abstraction level then it is a horizontal transformation. In MDA this can be either a PSM to PSM or PIM to PIM transformation. And when the source model and the target model both are located on the abstraction level that is at different levels then it known as vertical transformation. In MDA this is either a PSMs to code transformation or PIMs to PSMs transformation. [13]

Preliminary versions of MDA served for the reason that background for MDE, which usually generalizes the aspects of MDA, and defines the transformations inside context of metamodeling. Inside MDE, the most accepted strategy to define the products is through metamodeling strategies, and trans-formations tend to be defined using design transformation languages. On the other hand, in the first proposal of MDA, metamodeling had not been a necessary situation. It was simply in later versions of MDA that the ideas defined by means of MDE were incorporated, which led towards the definition of the MOF and QVT regular.

We can establish new metamodels using precisely the same tools used to be able to define models. MOF will be the basic vocabulary (called meta-metamodel) offered by the MDA standard to describe metamodels, and therefore brand new vocabularies (in fact we could say abstract 'languages', but we use the term metamodel in order to avoid confusion).

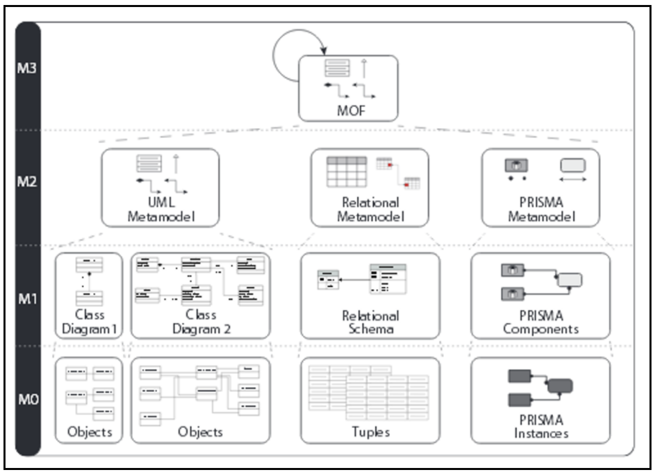

Fig 4: MOF Layers

However, one may wonder if there is a superior model vocabulary helpful to define meta-metamodels. But, as this artifact is additionally a model, could we carry on and expand this pyramid endlessly? The answer is actually yes, this style of meta-metamodels would be called meta-meta-metamodel.

In practice this makes simply no sense, and models as well as meta models tend to be organized into some sort of four-layer M3-M0 structure using the following distribution: [14]

It must be noted that this specific level usually has a unique vocabulary (meta-meta model) that characterizes the modeling tactic chosen is meta-metamodel have to be defined using its constructs, thus, the structure is closed in this particular layer. Level M3, which closes the structure towards the top, contains the basic vocabulary to spell out meta models. The syntax of the MOF language can be considered as a subset of the Unified Modeling Dialect class diagram. Within MOF, the meta-metamodel will be MOF itself. Metamodels, the languages to spell out new models, are simply in the decrease layer, the M2 coating. MOF–compliant metamodels are defined with regards to classes, attributes, associations, etc., which include the constructs provided through the meta-metamodel.

Use of MDA in Product line Engineering

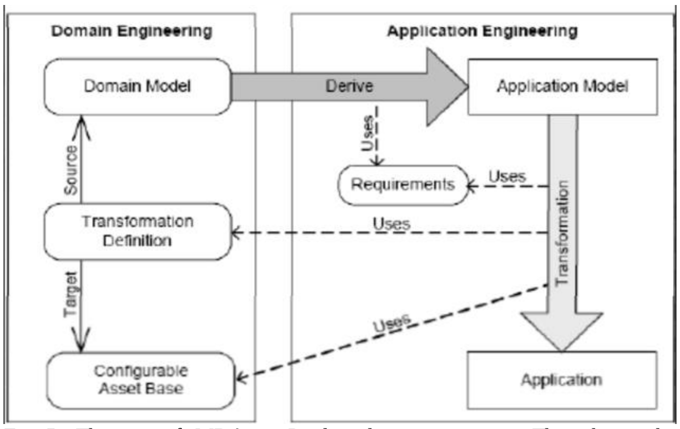

Fig 5: The use of MDA in Product line engineering. This shows the integration of MDA in SPL lifecycle.

Here, MDA is used as an approach to extract products in a specific type of SPL. The main aim here is to combine MDA with SPL and describe the advantages of this relationship and the main benefits identified were the independent evolution of domain concepts, the components of product line, and the management of variability.

The reuse and evolution by the use of MDA is supported by this method. According to the form view, it produces the architecture and product model with a standard notation (MDA) and in meta-types as level of abstraction. Architecture, product description, service and component are the nature of the derived product of this method. Instantiation is used as construction technique and its process covers the entire lifecycle. It finally tries to adapt operating conditions by the use of MDA which is capable of handling both platform and technical variability.

V. BENEFITS OF MDA

Technology obsolescence: New implementation infrastructure can be more easily integrated and supported by existing designs.

Portability: Existing functionality can be more rapidly migrated into new environments and platforms as dictated by the business needs. Only reconfiguration of the PSM and regeneration of the code is necessary to communicate with the new platform vendor.

Productivity and time-to-market: By automating many tedious development tasks architects and developers are freed up to focus their attention on the core logic of the system. MDA lets developers focus on the diagrams and describing text to regenerate code. Therefore, the artifacts are always up to date. As a result of which the project follow up is much easier.

Interoperability: Software systems rarely live in isolation. Most systems need to communicate often with already existing systems and it needs to get its information from existing back-end systems. MDA helps in interoperability by standardizing these communications.

Quality: The formal separation of concerns implied by this approach plus the consistency and reliability of the artifacts produced all contribute to the enhanced quality of the overall system.

Integration: The production of integration bridges with legacy and/or external systems is greatly facilitated.

Maintenance: The availability of the design in a machine-readable form gives analysts, developers and testers direct access to the specification of the system, simplifying their maintenance chores.

Testing and simulation: Models can be directly validated against requirements as well as tested against various infrastructures. They can also be used to simulate the behavior of the system under design.

Return on investment: Businesses are able to extract greater value out of their investments in tools.

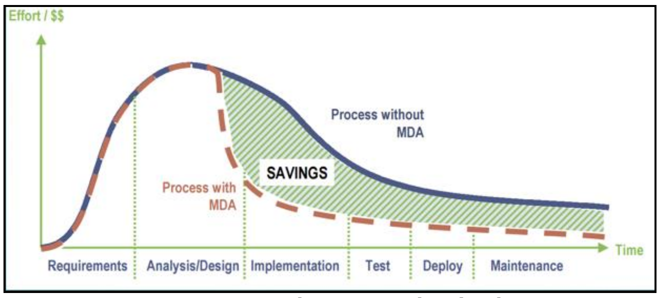

Fig 5: Comparative graph: Process with and without MDA

The above graph shows the comparison between two process i.e a process with MDA and a process without MDA with respect to effort and time. From the above analysis graph a process with MDA will considerably reduce the effort and time for various phases of software development.

VI. CASE STUDY

1) **PetStore:** It is a simple web-based J2EE e-commerce system. [15]

This is a case study by OMG showing the benefits of MDA related to increased developers productivity.

Two teams developed the same J2EE Pet Store application—one team used an MDA-based tool, while the other team used traditional approach.

The result of this study is the MDA team developed their application 35% faster than the traditional team. The MDA team finished in 330 hours compared to 507.5 hours for the traditional team.

Team	Original Estimated Hours	Actual Number Of Hours
Traditional Team	499	507.5
MDA Team	442	330

2) **UNISYS: [16]**

The UNISYS uses MDA approach. It uses the model based architecture in which more than 80% of the code is automatically generated from the models. The maintenance is done on the models, rather than the code. There would not be much changes required in code even when the requirement changes. This model-driven architecture (MDA) approach directly supports agility and flexibility.

Coding would take around 4 times longer without MDA, other activities (i.e. analysis etc) remain unchanged. Based on the survey conducted at UNISYS the following results were obtained. Without MDA (estd) $3 million in 77 weeks and With MDA (actuals) $1 million in 26 weeks.

Table 1: Below table represents the growing Impact of MDA in UNISYS

YEAR	2003	2005	2007	2009
MDA Penetration	Starts	10%	30%	60%
MDA-driven EAI			Starts	Common

3) IBM: [17]

IBM Rational Rose Technical Developer provides highly automated model transformation and robust code generation capabilities -- capabilities particularly important to developers of embedded systems and other technical software products. Similarly, IBM Rational Rapid Developer provides a highly automated implementation of MDA targeted to J2EE applications that integrate and extend existing legacy systems.

IBM strongly believes that organizations are well-served by creating models of the problem domain and solution domain, and by coordinating these models throughout the life of a software project. Because IBM has been a strong proponent of such model-driven approaches to software development, and model-driven development forms a key component of the best practices and tools available from IBM, today a wide range of IBM customers employ these technologies to great effect.

IBM has a long heritage of support for modeling, model driven development, and MDA is evident in many areas across IBM technologies and services. IBM Rational solutions in which modeling is used primarily to drive the design and implementation of enterprise-scale, software-intensive systems.

IBM Rational's leading role in defining the UML is widely acknowledged, as is the pre-eminence of the IBM Rational Rose product in implementing UML to support the architecting of large-scale software systems.

VII. CONCLUSION

Change is an inevitable and natural part of the software intensive system lifecycle. Changes can be made to extend assets or products, improve them or correct them. Mostly assets are dependent on other assets, and if one asset is changed this would have to change the related assets as well. On the other hand if changes made in PLAs assets this can have an effect on all the related assets.

SPL are coming out as a very viable and important paradigm allowing companies to realize order of magnitude improvements in cost, productivity, time to market and other business drivers.

The SPL approach, which, from a practical point of view is one of the most successful ones since it combines systematic development and reuse of assets, the products are different in some features but share a basic architecture.

The OMG, influenced from SPL has created a conceptual framework (MDA) that separates platform decisions from business-oriented decisions to allow greater flexibility when evolving and architecting these systems. The MDA framework is used as a blueprint by the application architects for expressing their enterprise architecture and employ the open standards inherent in MDA as their "future proofing" against vendor lock-in and technology churn.

The definition of a SPLs creation process is made possible by an approach which is a mixture of MDA and SPLs.

VIII. REFERENCES

Primary References:
Source: www.acm.org, www.computer.org

[1] Pierre F Tiako, Designing Software Intensive Systems : Methods and Principles

[2] www.omg.org/mda/faq_mda.htm

[3] Paul Clements, Linda Northrop, Software Product-lines: Practices and Patterns, AddisonWesley, ISBN 0201703327, August 20, 2001

[4] Dirk Muthig and Colin Atinkson, Model Driven Line Architectures

[5] Model Driven Development of Software Product Lines Alexandre Bragança1 and Ricardo J. Machado2(find conf. and year)

[6] A Model-Driven Approach for Software Product Lines Requirements Engineering Mauricio Alférez, Uirá Kulesza, André Sousa, João Santos, Ana Moreira, João Araújo, Vasco Amaral

[7] Addressing Domain Evolution Challenges in Model-Driven Software Product-line Architectures Gan Deng , Gunther Lenz , Douglas C. Schmidt

[8] 16 Evaluation Framework for Model-Driven Product Line Engineering Tools J. Oldevik, A. Solberg , Ø. Haugen, and B. Møller-Pedersen

[9] Variability Modelling for Model-Driven Development of Software Product Lines∗ Ina Schaefer

[10] Vijaya Devi (2013), Traditional and Agile Methods: An Interpretation

[11] Muthig, D., & Atkinson, C. (2002). Model-driven product line architectures. In Software Product Lines (pp. 110-129). Springer Berlin Heidelberg

[12] Greenfield, J., Short, K., Cook, S., Kent, S., & Crupi, J. (2004). Software factories: assembling applications with patterns, models, frameworks, and tools (pp. 16-27). Wiley Pub..

[13] Voelter, M., & Groher, I. (2007, September). Product line implementation using aspect-oriented and model-driven software development. In Software Product Line Conference, 2007. SPLC 2007. 11th International (pp. 233-242). IEEE.

[14] Deelstra, S., Sinnema, M., Van Gurp, J., & Bosch, J. (2003, June). Model driven architecture as approach to manage variability in software product families. In Proceedings of the Workshop on Model Driven Architectures: Foundations and Applications (pp. 109-114).

[15] The Middleware Company : Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach

[16] David M Bridgeland (2003) Unisys Business Blueprints : Model Driven Architecture and Enterprise Integration

[17] Alan Brown (2004) An introduction to Model Driven Architecture.