ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 1, Issue 10, December 2012

Xml Based Solution to MDA

Mr. Ketan B. Rathod
M.E(3rd Sem) CSE
Parul Institute of Technology
Vadodara-India

Ms. Astha Baxi
Assistant Professor, CSE Dept.,
Parul Institute of Engineering and Technology
Vadodara-India

Abstract: **We know that UML is widely used for the specification and modeling of software. Model driven approach uses unified modeling language as platform independent model and convert it into platform specific model by adopting different strategies. However nonuniformity in strategy makes UML based MDA a challenging job. Also very less number of platform specific code is generated when UML Platform Independent Model is converted to Platform Specfic Model. Until now we are having a tool which can generate a code in JAVA & php language. We are proposing a tool which is using XML as PIM and generating the code in Php,C# & C++ languages, and less manual coding. It reduces the software cost and enables the user to change their platform in intermediate processing of the tool without manpower work.So, can generate similar projects in less effort on different platforms. Web services are also an integral part of this paper.UML does not allow web services and generation of codes in multiple languages from a given single tool, which the latterly XML do provide us. XML provides uniformity in description of different component.**

*Keywords:*Code Generation, Software Engineering, UML, XML

INTRODUCTION

Software Engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches; that is, the application of engineering to software. It is the application of engineering to software because it integrates significant mathematics, computer science and practices whose origins are in engineering. It is also defined as a systematic approach to the analysis, design, assessment, implementation, testing, maintenance and re-engineering of software, that is, the application of engineering to software.

SDLC-A Software development process, also known as a software development life cycle (SDLC), is a structure imposed on the development of a software product. Similar terms include software life cycle and software process. It is often considered a subset of systems development life cycle. There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process. Some people consider a life-cycle model a more general term and a software development process a more specific term. For example, there are many specific software development processes that 'fit' the spiral life-cycle model. ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

SOFTWARE DEVELOPMENT ACTIVITIES

Software development is achieved through a collection of activities that lead to the building of a software product. Software development is traditionally divided into various phases, which we describe briefly in the following sections. For small projects, the phases are carried out in the order shown; for larger projects, the phases are interleaved or even applied repetitively with gradually increasing refinement . The strict definition and organization of these phases is called a software process.

REQUIREMENTS ANALYSIS

Extracting the requirements of a desired software product is the first task creating it. While customers probably believe they know what the software is to do, it may require skill and experience in software engineering to recognize incomplete, ambiguous or contradictory requirements. Requirements analysis can itself be broken down in sub-activities. This phase is often the topic of processes itself, often referred to as the requirements process, or even requirements engineering process, the latter title being debatable. Requirements analysis methodologies have been created, one of the most popular being the use case driven methodology.

SPECIFICATION

Specification is the task of precisely describing the problem description, often in a mathematically rigorous way, in most cases actually building a more or less complete model of the problem to be solved. A wide array of specification techniques exist, and many application areas rely on dedicated specification techniques. In practice, most successful specifications are written to understand and fine-tune applications that were already well-developed. Specifications are most important for external interfaces, that must remain stable. This is often referred to as software interface specifications. Note that internal software interfaces are generally established in the design phase of development, which makes this kind of specification a design artifact. However, some systems have external software interfaces with other existing systems. The specification of these interfaces then becomes something that is very important to specify in the early phases of development, and is thus really considered a specification artifact.

DESIGN

Design refer to conceptually determining how the software is to function in a general way without being involved in low-level operational details.
Usually this phase is divided into two sub-phases, such as architectural design, detailed design, or algorithmic design. Design can also be categorized into different focus such as graphical user interface (GUI) design, or database design, depending on the nature of the application. Design is an extremely important phase in the development of the software, as it permits the developers to think in relatively abstract terms about the solution. This is in contrast with the implementation phase, where the solution is approached from a very concrete and often short-sighted point of view. Approaching the problem from an abstract point of view permits to 'see the big picture', and develop abstract models of the solution that can be easily modified as details are grafted and the solution become more and more concrete. At a certain point, the designed solution will be concrete enough so that the implementation (i.e. coding) phase can start.

IMPLEMENTATION

Reducing a design to code may be the most obvious part of the software engineering job, but it is not necessarily the largest portion. A common point in software development practice is that coding should be the main focus in software development. Such a misconception is a popular reason for low software quality or software project failure. Note that effective programmers have to deal with quality factors, and provide qualities such as code readability, maintainability, and often test the code as they are writing it.

TESTING

Another common fallacy of software development is that testing is mostly about executing the application and making sure that it does not crash before it is made operational. Testing is basically about verifying the quality of the product, and validate that it meets its requirements. In fact, "testing" is about the validation and verification of the various artifacts produced during the development of the software. Not only the code.
For example, when requirements are established, one has to assess their quality in order to prevent that further phases be implementing a faulty or incomplete solution. The keyword "Testing" is most often referring to quality assurance of the code itself, but real software engineering will care about the quality of all artifacts produced, not only the code or the final application.

MAINTENANCE

Maintaining and enhancing software to cope with newly discovered problems or new requirements can take far more effort than the initial development of the software. Not only may it be necessary to add code that does not fit the original design but just determining how software works at some point after it is completed may require significant effort. About 2/3 of all software engineering work is maintenance, but this statistic can be misleading. A small part of that is fixing bugs. Most maintenance is extending systems to do new things, or adapting it to a new environment, which in many ways can be considered new work. Similarly, about 2/3 of all civil engineering, architecture, and construction work is maintenance.

MODEL DRIVEN ARCHITECTURE

Model-Driven Development (MDD) claims that software systems must be developed through the use of models. MDD processes usually start to develop a software system with a requirements phase in which a requirements model is defined to describe the user's needs in a computation-independent way. Then, this model is refined into one or more conceptual models that describe the system without considering technological aspects. These conceptual models are focused to be used in analysis phases. Finally, these models are (1) either refined into design models that describe the system by using concepts of a specific technology and are then translated into code, or (2) directly derived to code if they contain enough information to implement the software product in a precise and complete way. The Model-Driven Architecture starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. MDA provides an approach for, and enables tools to be provided for:
-Specifying a system independently of the platform that supports it
-Specifying platforms
-choosing a particular platform for the system, and
-transforming the system specification into one for a particular platform The three primary goals of MDA are portability, interoperability and re-usability through architectural separation of concerns.

PLATFORM INDEPENDENCY

Platform independence is a quality, which a model may exhibit. This is the quality that the model is independent of the features of a platform of any particular type. Like most qualities, platform independence is a matter of degree. So, one model might only assume availability of features of a very general type of platform, such as remote invocation, while another model might assume the availability a particular set of tools for the CORBA platform. Likewise, one model might assume the availability of one feature of a particular type of platform, while another model might be fully committed to that type of platform. The MDA is a new way of developing applications and writing specifications, based on a **platform-independent model (PIM)** of the application or specification's business functionality and behavior.

PLATFORM SPECIFIC MODEL (PSM)

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.

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.

IMPLEMENTATION

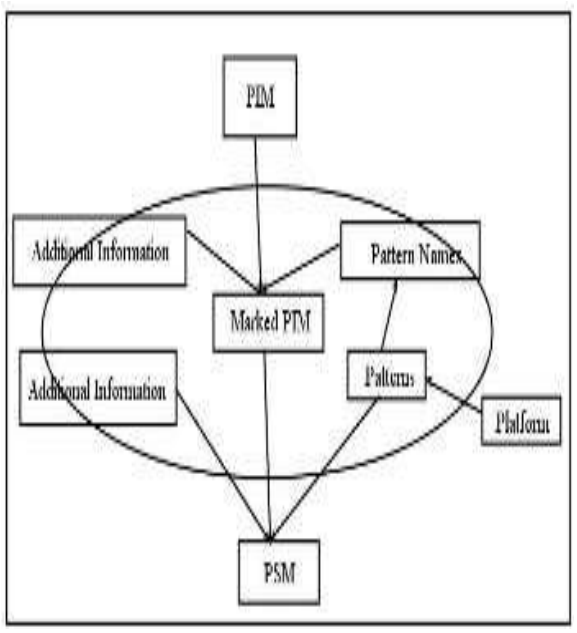

Fig.1 MDA

Currently UML is used as de facto standard of generating PIM. The following are the advantages of using UML as PIM.

1.Use of Model: Models are visual components to generate template. To describe something visually gives great flexibility in understanding and designing.

2.Re-usability: Models can be used to generate similar project again and again.

3.Testing: Testing on template can be done on early stage of designing. So it becomes easy to detect some syntactical error at very early stage.

4.Code Density: Using UML total code to the extent of 30-35% can be generated. This percentage reduces a great effort of programmer.

The introduction of UML as PIM language changes the SDLC and now some codes are generated and tested automatically at early design stage. Due to features showed by UML as PIM many scholars talked about UML and proposed different kind of strategies of using UML for generating different kind of system.

DISADVANTAGE OF UML

Interoperability : There are many software available which supports MDA core functionality. But they stores information in different format.

Require high bandwidth for network transmission : UML diagram is stored/saved in different binary format. In a large enterprise application this diagram become huge in size.

Lack of templatization at method/function level.

Unable to generate fix syntax code.

Solution:

Use XML instead of UML so that the fixed syntax logic generates the larger number of codes which is been pre-tested, that it, the code which is not to be tested. Thereby, Decreases the manual coding work and subsequently the software cost also get reduced. Below figure shows the flowchart of how exactly the XML works and what all are the steps which is to be taken care of. A tool is designed on java platform which gives the following result.

- It is a menu driven software which generates XML for given software.
- Currently It generates Classes, Function Calling & Database Connection code.

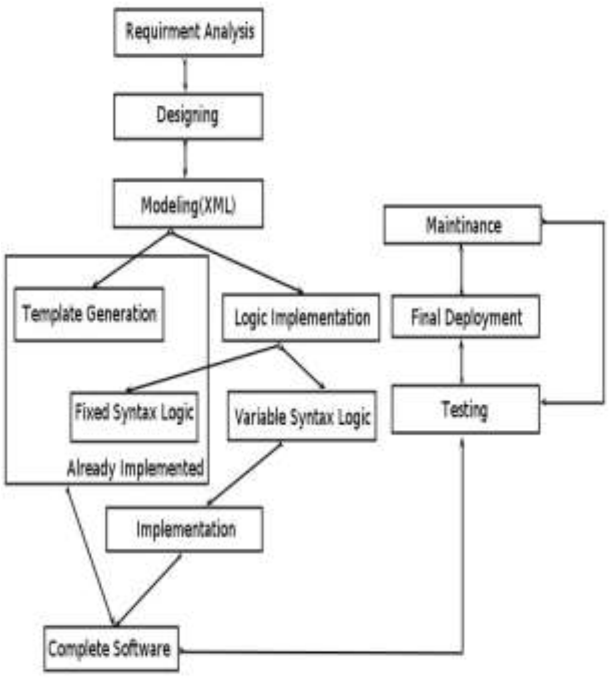

Fig.2 flowchart of solution

XML can be use as we can define our own tags in it and also it produces the large amount of codes as compare to UML. By using this we can easily switch on to different languages if the client changes his mind. So, there is less manual coding is required, and also the software cost is being reduce.

Why XML?

In order to appreciate XML, it is important to understand why it was created. XML was created so that richly structured documents could be used over the web. The only viable alternatives, HTML and SGML, are not practical for this purpose.

HTML, as we’ve already discussed, comes bound with a set of semantics and does not provide arbitrary structure. SGML provides arbitrary structure, but is too difficult to implement just for a web browser. Full SGML systems solve large, complex problems that justify their expense.

Viewing structured documents sent over the web rarely carries such justification. This is not to say that XML can be expected to replace SGML. While XML is being designed to deliver structured content over the web, some of the very features it lacks to make this practical, make SGML a much more satisfactory solution for the creation and long-time storage of complex documents. In many organizations, filtering SGML to XML to serve it over the web will be standard procedure.

XML DEVELOPMENT GOALS

1.It shall be straightforward to use XML over the Internet. Users must be able to view XML documents as quickly and easily as HTML documents. In practice, this will only be possible when XML browsers are as robust and widely available as HTML browsers, but the principle remains.

2.XML shall support a wide variety of applications. XML should be beneficial to a wide variety of diverse applications: authoring, browsing, content analysis, etc. Although the initial focus is on serving structured documents over the web, it is not meant to narrowly define XML.

3.XML shall be compatible with SGML. Most of the people involved in the XML effort come from organizations that have a large, in some cases staggering, amount of material in SGML. XML was designed pragmatically, to be compatible with existing standards such as SGML while solving the relatively new problem of sending richly structured documents over the web.

4.It shall be easy to write programs that process XML documents. The colloquial way of expressing this goal is that it ought to take about two weeks for a competent computer science graduate student to build a program that can process XML documents.

5.The number of optional features in XML is to be kept to an absolute minimum, ideally zero. Optional features inevitably raise compatibility problems when users want to share documents and sometimes lead to confusion and frustration.

6.XML documents should be human-legible and reasonably clear. If you don’t have an XML browser and you’ve received a hunk of XML from somewhere, you ought to be able to look at it in your favorite text editor and actually figure out what the content means.

CONCLUSION

Using UML as model gives a great way to represent and generate a project, but as a diagram it can not describe each and every component of a software system. Components can be identified by adopting different approach for different component. UML provides no generalized syntax for different kind of component. UML based modeling lacking behind with respect to current pace of software industries as it generates less number of executable code. Testing a maually generated code consumes a lot of time which result in software deployment delay. The main disadvantage of UML is that it does not recognize the fixed syntax code. The more fix syntax code will be recognize the more executable code can generate automatically and lesser amount of time will be devoted to testing. This strategy will speed up the production rate and unusual delay can be minimized. The limitation of UML can be overcome if we use XML. Being a narrative language, a lot can be described about a project in terms of tags and attributes of XML. we can describe about classes, methods, aspects etc by designing tag. There are two major problems which are still in existence and requires extensive effort to solve. These problems are Reducing cost of software and Generating similar projects in less effort.

FUTURE WORK

Currently the parser is limited to generating code in two languages JAVA & php. Future work will include an implementation of a tool which will provide a complete visual aspect in all platform and having Web services in it.

REFERENCES

[1] Xml a way to maximize the code generation at industrial level by Atul Saurabh.

[2] Torsten Lodderstedt, David Basin, Jurgen Doser: SecureUML: An UML Based Modelling Language for Model-Driven Security, publisher Springer, doi 10.1007/3-540-45800- X_33 2002

[3] Fabian Mischkalla, Da He, Wolfgang Muller : Closing the gap between UML-based Modelling, Simulation and Synthesis of combined HW/SW system, publisher IEEE Computer Society pno : 1201 - 1206 , 2010

[4] Pedro J. Clemente, Juan Hernandez, Fernando Sanchez: An MDA Approach to Develop Systems Based on Components and Aspects, published in ACM New York, NY, USA, pno : 1033 - 1034, doi :

[5] Matteo Golfarelli, Stefano Rizzi: UMLBased Modeling for what-if analysis, published in Springer-verlag Berlin Heidelberg doi: 10.1007/978-3-540- 85836-2_1

[6] Pekka Aho, Matti Maki, Daniel Pakkala, Eila Ovaska: MDA-Based Tool Chain forWeb Services Development, publisher ACM New York, NY, USA, pno : 11-18, doi : 10.1145/1645406.1645409 2009

[7] Belen Vela,Eduardo FernÃ¡ndez- Medina, Esperanza Marcos,Mario Piattini: Model Driven Development of Secure XML Databases SIGMOD Record, Vol. 35, No. 3, Sep. 2006

[8] Joerg Evermann, Adrian Fiech, Farhana Eva Alam : A Platform- Independent UML Profile for Aspect-Oriented Development, published in ACM New York, NY, USA pno: 143-147, doi : 10.1145/1557626.1557649

[9] Object Management Group: OMG Unified Modeling Language Specification 2.4.1 2012, http://www.omg.org/spec/UML/2.4.1/, accesses on November 2011

[10] World wide web consortium : http://www.w3.org,

[11] Object Management Group, 2010. UML Resource Page. http://www.omg.org/uml/

[12] Zhongjie Wang, Xiaofei Xu, Dechen Zhan.: A Survey of Business Component Identification Methods and Related Techniques. In: International Journal of Information Technology, vol. 2 (2005)

[13] Devon Simmonds, et al.: An Aspect Oriented Model Driven Framework. In: Proc. of the 9th IEEE International EDOC Enterprise Computing Conference (2005)

[14] http://en.wikipedia.org/wiki/MDA

[15] http://www.w3.org/TR/xpath/

[16] http://www.w3.org/XML/