Building Domain Specific Enterprise Applications using Model Driven Development

Clarence J M Tauro
Centre for Research, Christ University
Hosur Road, Bangalore, India

N Ganesan
Director (MCA), RICM
Bangalore, India

Vijay Gopal.M

Rinu Thomas

ABSTRACT
In this paper we explain an approach on how to develop domain specific applications using model driven development. Model Driven Development in its core, uses the MDA (Model Driven Architecture) principles defined by OMG (Object Management Group) and its primary artifact for development is model. MDA principles dictate that the domain specific model be built on specifications and standards [1].
On the other hand Domain Driven Design principles addresses the domain problem in a well defined manner that when captured as requirement and developed as a system results in a cohesive system that addresses the business problem [2].

Our discussion previews the Domain Driven Design principles for developing a domain specific application, limitations of traditional software development and highlights the advantages of Model Driven Development and an example explaining the discussed principles.

Keywords
Model Driven Development, MDD, Domain Driven Design, CQS, UML

1. INTRODUCTION
The current software development methodologies have passed its infancy stage. The advent of new software tools and techniques to ease software development process has played vital role in the area of software development and delivering solutions on time. In addition to the evolving software methodologies the business domains have also evolved and turned complex which poses serious challenges in designing software. Rapidly changing requirements and stringent deadlines, high quality and reliability of the software gets challenged due to faster delivery mechanisms. Ensuring reduced project costs and time to market has actually hit the software quality and reliability and in turn increases costs.

In traditional software development both the domain model and the technology are tightly coupled. The rapidly changing requirements dictate the domain model to change accordingly and provide the desired outcome. Few domains dictate that the errata must be negligible For Example: Healthcare, then there are more likely chances that the project will fail. There might be other issues such as the developer or the business analyst himself is not able to understand the problem statement in sync with the current system then it might result in the development of a system which will eventually lead to the failure of the project.

The challenge in developing such rapidly changing software is the mapping of the domain to the constantly developing solution. Successful software is one which captures the business requirement in the context of the stakeholder, transform the same business problem without any loss of business understanding as technical requirement for the developer and at last develop a solution which addresses the business problem that meets the stakeholder's expectation.

The core idea of this paper is to provide a design solution in order to develop a domain specific enterprise application. This paper consists of five parts. Principles of Domain Driven Design, Issues in traditional software development, Need for Model driven development, Sample Solution and Conclusion and Future Work.

2. PRINCIPLES OF DOMAIN DRIVEN DESIGN
Domain Driven Design's primary principle is to understand the deep issues of the domain collaborate with the domain experts and design a conceptual model to build a software solution that addresses the domain problem. It is a paradigm shift in designing software where the core focuses is on the problem bounded by the domain. Domain Driven Design is best expressed by using modeling languages [8]. Modeling languages need not necessarily be UML. They could be simple whiteboard diagrams as well. In order to have a consistent understanding among the developers, the industry specifications standard such as UML is followed. A core principle of Domain Driven Design is to use a language which is understood by all stakeholders. Hence domain driven languages are ubiquitous by nature [3]. Such languages provide consistent understanding across the project lifecycle. Hence there is no loss of knowledge during the initial phases of software development.

The core idea of Domain Driven Design is not to bind the domain to any implementation, rather express the domain problem in such a way that the designers and architects design the artifacts that represent's the domain. The application model is usually large to comprehend as a whole. Hence the Domain Driven Design is always bounded by a context called "Bounded Context". Bounded context is specific to each subpart of the main model [3]. For Example: Shopping cart context allows the user to select items, remove the items, search the catalogue and display the cart items. This helps us to have a clear segregation of responsibility. Bounded Contexts has a well defined responsibility and must be able to communicate with other bounded contexts.

Domain Driven Designs another core principle is the CQS (Command Query Separation) which is used in imperative computer programming and borrowed from Eiffel programming language. This pattern states that "every method should either be a command that performs an action or a query that returns data to the caller, but not both". But this is an implementation level detail. Bringing this detail to a layer up addresses the reporting issues found in traditional applications. Hence we use the command for domain operations and query for reporting operations.

The core elements of Domain Driven Design are the value objects and entities. An entity is unique in the system and has an identity. A value object in turn doesn't have an identity and

describes itself with the help of attributes and a value object cannot live on its own. Hence it must be immutable.

The final element of Domain Driven Design is the Aggregates and Aggregate Root. Aggregates are collection of entities and value objects. But an aggregate root must always be an entity [9].

Using the above principles in the area of software design can result in well designed software with less documentation effort.

3. ISSUES IN TRADITIONAL SOFTWARE DEVELOPMENT

Software development is inherently complex by nature. Ensuring a software delivery with expected quality has become difficult to achieve due two major factors.

1.Technical Factors

2.Business Factors

3.Domain Factors

The technical factors such as interoperability of the systems, rapidly changing technology, upgrading from older to newer systems and custom application development often throws unseen challenges. Most of the time it throws challenges in terms of understanding the newer technology stacks and so on. Though less it throws its own share of hurdles in the area of software development.

Business Factors contributes as the prime factor in the area of software development. Business plays the vital role in the area of software development. The business factors contribute to the area of software development are time to delivery and time to market, stringent business processes and evolving business standards. For Example: The offshore onsite delivery model which is focused at reducing costs and faster time to market actually at times increase cost due to reduced development time and poor quality of development. Most of the time, it becomes difficult to maintain piles of documentation for every change done in software. During the estimation the time for this activity is never taken and the developer is burdened. Hence loses focus on the deliverable and works on activities which surround the deliverable. Sometimes due to reduced market time organizations resort to COTS (Commercial of the Shelf). It does provide a solution and along with it comes its own set of problems, such as interoperability and closed source code. Though these problems are not directly related to the development activity yet these balks the progress of software development.

The key challenge in building the enterprise application is to understand the problem domain [5]. Most of the time the requirements are captured either too broadly or in a shrewd manner. This results in loss of actual problem information. Even the experts fail sometimes in understanding the requirement. Or it also happens that the client himself is not clear of what he wants from the software. And above that the requirements tend to change over time. Mapping these changing requirements to the developing software and ensuring that these requirements are in alignment with the domain is one critical challenge.

These challenges balk the progress of the software development as a whole. Addressing these challenges from the beginning could help us to deliver successful software.

4. NEED FOR MODEL DRIVEN DEVELOPMENT

Model Driven Development is all about developing software with the help of software models. Model Driven Development in its core follows MDA (Model Driven Architecture) principles. With the help of modeling one can translate the business to much low level technical detail. UML is the de-facto used in Model Driven Development due to its wide acceptance [4].

Modeling is the language of the business analysts. Models help to bridge the gap between gathering requirements and translation of the same requirements to technical requirement [6]. These models when used with specific tools generate code which adheres to architectural principles [7]. Most of the time due to stringent deadlines or lack of understanding of architectural design patterns developers might try to breach the design principles and develop software that are unstable or prone to enhancement and maintenance issues.

Amidst many development methodologies why do we need model driven development? The current buzz word in software industry is Agile Development methodology. Agile development is a discipline, the ability to adapt to changing needs of the customer [8]. This methodology is customer driven and has better principles such as better return on investment, improved software quality and improved control on the development activity due to constant interaction with the client.

The aim of Model driven development is to transform the model to code and during this phase their is greater chance of errata due to wrong interpretation of the model. By using sophisticated tools the model can be used to generate code , which is basically the skeletal of the system and the developer is focused to code only the business logic of the system. This results in improved productivity.

By incorporating agile principles with MDD we can address various issues of software development from technical and business perspectives. For Example: One can achieve improved quality by constantly interacting with the client and incorporating the suggested changes in the developing software and showcasing the same to the client. The time taken to incorporate a change and see the immediate result is extremely less when compared to traditional methodology. This helps in understanding the changing software needs, develop software with acceptable quality and overall better returns for the organization and better understanding of the domain. Hence we consider MDD as a pervasive development methodology in the area software development which addresses the issues in traditional software development.

4.1.Sample implementation

Consider the code in Figure 1

```
public class public TradingService {
public Money
calculateOrderEstimate(Account account,
Order order){

// Calculate basic order estimate
Money unitPrice;
if (order.getType()==orderType.Market) {
unitPrice=
marketDataService.getMarketPrice(order.getS
ymbol());
}
else
{
unitPrice = order.getLimitPrice();
}

Money estimate =
unitPrice.times(order.getQuantity());

// Calculate fees
Money fees =
(account.getOwner().isPreferred()) ?

DISCOUNTED_FEES : REGULAR_FEES;
// Return estimate plus fees
```

Figure 1: Anemic Code

The code in Figure 1has anaemic problems .This program does a lot of tasks at the same time. Hence we use the first basic principle of Domain Driven Design CQS (Command Query Separation). The other issue associated with the above code in Figure 1 is the maintainability issue.

```
public class TradingService {
public Money calculateOrderEstimate(
Long accountId, OrderParams orderParams)
{
Account account=
accountRepository.findAccount(accountId)
;
return account.calculateOrderEstimate
(new Order(orderParams));
}
```

Figure 2: Trading Service

```
public class Account {
public Money calculateOrderEstimate(Order
order)
{
Money estimate = order.estimate();
Money fees = this.isEligibleForDiscount() ?
DISCOUNTED_FEES : REGULAR_FEES;
return estimate.plus(fees);
}
}
```

Figure 3: Account Class

```
public class Order {
public Money estimate() {
Money unitPrice;
if (this.type == OrderType.Market)
unitPrice =
marketDataService.getMarketPrice
(this.symbol);
else
unitPrice = this.limitPrice;
return unitPrice.times(this.quantity);
}
}
```

Figure 4: Order Class

Figure 2, 3, 4 discusses about the implementation of CQS design pattern where every business logic is separated as per the bounded context principle. This also clearly signifies the separation between the validation logic and the business logic. Now the separated logic has to be placed in different layers as per the layering design principle. An enterprise application usually consists of different layers. Apart from placing the segregated logic in different layers we have to ensure that the aggregate principle is also implemented. Given below are the different layers for a domain specific enterprise application.

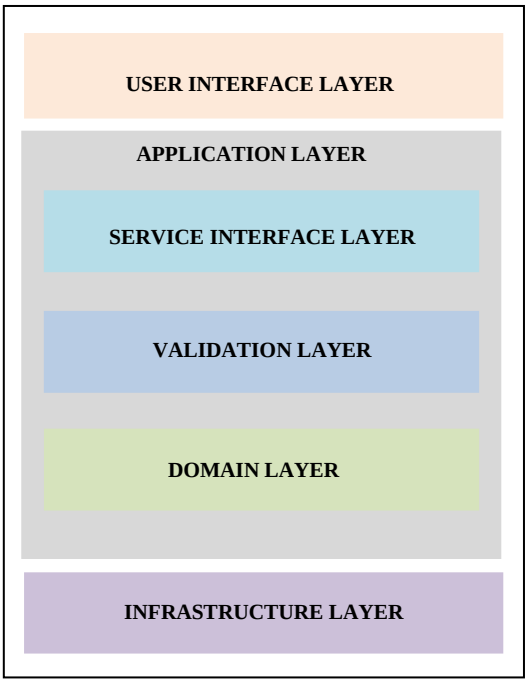

Figure 5: Architecture of Domain Driven Design

The Figure 5 describes the layering architecture used in Domain Driven Design. The architecture clearly describes the segregation between the business logic, the infrastructure, the services and the validation parts. The rules layer is optional and contains the rules pertaining to specific domain. Ex: Government services domain.

5. CONCLUSION AND FUTURE WORK

This paper illustrates various issues found in the traditional software development. Our study is categorized only to the area of capturing requirements in alignment with domain, development of design in alignment with domain principles and realizing the design in the code with model driven

development. These challenges have been summarized by prior literatures. The major challenge in incorporating the suggested solution is to identify the right problem statement where the domain complexity is extremely high. It also includes a paradigm shift in the thought process of the developers and designers.

In future we will be looking forward to incorporate the suggested solution to those domains which have intrinsically complicated problem statement which are difficult to solve by traditional development methodology.

6. ACKNOWLEDGMENT

We are heartily thankful to Prof. Jibrael Jos and Prof. Joy Paulose, Department of Computer Science, Christ University, whose encouragement, guidance and support enabled us to develop an understanding of the subject.

7. REFERENCES

[1] Marzullo, F.P.; de Souza, J.M.; Blaschek, J.R.; , "A Domain-Driven Development Approach for Enterprise Applications, Using MDA, SOA and Web Services," E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services, 2008 10th IEEE Conference on , vol., no., pp.432-437, 21-24 July 2008 doi: 10.1109/CECandEEE.2008.119URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4785103&isnumber=4785030

[2] Laufer, K.; , "A Stroll through Domain-Driven Development with Naked Objects," Computing in Science & Engineering , vol.10, no.3, pp.76-83, May-June 2008

doi: 10.1109/MCSE.2008.67 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4488069&isnumber=4488052

[3] Abel Avram & Floyd Marinescu, "Domain-Driven Design"

[4] Trask, B.; Roman, A.; "Leveraging Model Driven Engineering in Software Product Line Architectures," Software Product Line Conference (SPLC), 2011 15th International , vol., no., pp.356-357, 22-26 Aug. 2011

doi: 10.1109/SPLC.2011.63 URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6030091&isnumber=6030038

[5] Jamaludin Sallim , "Requirements Engineering for Enterprise Applications Development: Seven Challenges in Higher Education Environment," World Academy of Science, Engineering and Technology 4 2005.

[6] Gholami, M.F.; Ramsin, R.; , "Strategies for Improving MDA-Based Development Processes," Intelligent Systems, Modelling and Simulation (ISMS), 2010 International Conference on , vol., no., pp.152-157, 27-29 Jan. 2010 doi:10.1109/ISMS.2010.38

[7] Jicheng Fu; Wei Hao; Bastani, F.B.; I-Ling Yen; , "Model-Driven Development: Where Does the Code Come From?," Semantic Computing (ICSC), 2011 Fifth IEEE International Conference on , vol., no., pp.255-262, 18-21 Sept. 2011 doi:10.1109/ICSC.2011.76

[8] Aniket Mahanti." Challenges in Enterprise Adoption of Agile Methods – A Survey," Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on , Journal of Computing and Information Technology – CIT 14, 2006, 3, 197–206 doi:10.2498/cit.2006.03.03