Domain Driven Architecture for Domain Experts Knowledge Representation in Software Design and Development

Kalashankar & N.N.S.R.K. Prasad

IT Consultant, Logica, 124-125, Divyasree Technopolis, Yemlur, Bangalore - 560 037, India
Group Director (Electromagnetic & Optical Systems), ADA, PO:1718, Vimanapura Post, Bangalore-560017, India

E-mail : kba.shetty@gmail.com, nnssrkprasad2007@gmail.com

Abstract – In order to represent, understand the complex software projects in terms of customer words is only through Domains Model Architecture. The domain model speaks about customer needs, essentials; requirements and it also contribute to develop the system with more robust, reliable and understandable with less cost and less complexity. This can be achievable through Model Driven Architecture (MDA) and Model Driven Software Development initiatives, have brought back the idea of generating artifacts from models utilizing Domain Specific Languages and are constantly gaining momentum in industrial environments.

The IT Industry has been extensively contributing in diversified approach, so to talk; real challenges of complex software projects is not technical realization, but the adequate conversion of the business process itself, knowledge representation of domain expert and the demands of the professional user. The Domain Driven Design (DDD) realm has proven to be very helpful in constructing healthy and sustainable software [1]. In this paper we compare and discuss the domain model components, definitions, domain expert knowledge representation, domain layer architecture and considering a detail study on one particular domain architecture to explain the complexity while using the domain entities.

Keywords – Agile Model, Domain Objects, EMF, Satellite Domain, Scrum Model, UML Model.

I. INTRODUCTION

The Domain Models in Software Development

The domain model is essential in software design and development; it reflects the understanding of a particular domain using classes, attributes, relationships and collaboration behavior. It can be extended with additional information and constraints. The domain model is the central communication tool which constantly nourished with further insights and requirements. The domain model classes need very little framework support. They are implemented as Plain Old Java Objects (POJO’s). The business rule implementations do rely on some classes of the Spring framework, but they are implemented as separate classes. The domain model classes are not in any way dependent on them. In other words the heart of the DDD is Model and we start developing our application with drawing our model. Model and Design we create should shape each other. Model should represent knowledge to the business and it is language our team speaks.

Focus on the Domain Model

The designers and developers can finally focus on the domain model. They design the technical domain model; they add hand-written code supplementing the resulting generated code. All the rest – the layered architecture, the technologies used, etc. is taken care of by the generator provided.

II. LITERATURE SURVEY

Ubiquitous Language representation for satellite domain

Consider a scenario, where a satellite domain Expert, talking to the developer or with development team. How does that conversion look like? It could be out of the box and this creates serious problems to our
project. Domain Experts have their own jargon and the development team has its own. The daily discussions with Domain Experts and the team are represented the terminology embedded into code. So effect of need to translate what you are talking about into code appears and also the misunderstandings while getting translation. This all could lead to not good results due to not having the common language.

So there is a need for some common language shows up. Eric Evans calls this language as Ubiquitous Language[1]. Your language is your model; this means that both Domain Experts and Developers should talk the same language which is build on the model. And changes to the Ubiquitous Language means changes to the Model. If Experts start using new terms this should be represented in our model, in our code, diagrams and speech.

Domain-Specific Language

Domain-specific language is formal language used to describe the business and professional tasks and fields of responsibility within the domain. Independent of the programming language, its purpose is to express client requirements and the software functions that are designed to meet them in the user's own terminology. This enables a more intensive and efficient exchange between domain experts, users and developers.

The most important advantages of domain-driven design are the demand-oriented presentation of business and professional logic (Domain model) within a user-oriented Common Language (domain-specific language) [6]. These result in faster analysis, design and development processes and, above all, in software solutions that are entirely appropriate to customer requirements. Such solutions give an adequate representation of the client's needs in details.

Case study - 01:
Satellite Domain Knowledge Representation

When it comes to the Satellite Domain, this domain knowledge represents very high level complexity for end users. The people who are from telecommunication back ground can capture the terms easily and understand the Domain flow; if the developer is from the computer science background then it will be difficult while designing the Domain Model. The fundamental components shown in below figure, the design of the overall system determines the complexity of the various components and the manner in which the system operates. The basic design of a satellite Domain system depends to a great degree upon the characteristics of the orbit of the satellite and its behavior.

Satellite Communication

The satellite communications has demonstrated that satellite systems can satisfy many requirements. They are reliable, survivable, secure, and a cost effective method of telecommunications. We can easily see that satellites are the ideal, if not often the only, solution to problems of communicating with highly mobile forces. Satellites, if properly used, provide much needed options to large, fixed-ground installations.

Communications via satellite is a natural outgrowth of modern technology and of the continuing demand for greater capacity and higher quality in communications. The below figure represent the Satellite Domain Tree view structure having each aggregate with its boundaries (just overview):

Documentation

Documentation is a continuous effort. We have to document the model using free-text notes as we design it, we document the code as your write it. Documentation – in the sense of Eric Evans always uses the ubiquitous language. Further documentation describes the APIs including the services that are exposing your domain model towards a client.

Code

The behavior of objects can be best expressed with code. We basically do not put any code in the model, but we nourish the generated code with additional functionality. The initial result from a generator run on the constructed model can therefore only represent the preliminary object model. The specific behavior has to be added manually by the software developers.

Modeling is an Art - UML Model

From the inputs and outputs of the Business Process Model and the details of the use cases, begin to construct a domain model (high level business objects), sequence diagrams, collaboration diagrams and user interface models [2]. These describe the ‘things’ in the new system, the way those things interact and the
interface a user will use to execute use case scenarios. From the domain model, the user interface model and the scenario diagrams create the Class Model. This is a precise specification of the objects in the system, their data or attributes and their behaviour or operations. Domain objects may be abstracted into class hierarchies using inheritance.

### Agile Model

Agile software development permits the writing, embedding and testing of new or modified functional requirements at any time, even during the final phases of a project. Nowadays Agile is a way to develop software and DDD is very comfortable with it and with techniques of the Agile like TDD.

Agile software development teams embrace change, accepting the idea that requirements will evolve throughout a project [4]. The Agile specialist understand that because requirements evolve over time that any early investment in detailed documentation will only be wasted. Instead they will do just enough initial modeling to identify their project scope and develop a high-level schedule and estimate; that’s all you really need early in a project, so that’s all we should do.

### Scrum Model

Scrum suggests that you freeze the requirements for the current iteration to provide a level of stability for the developers. If you do this then any change to a requirement you’re currently implementing should be treated as just another new requirement.

### Domain Model

The domain model is an aggregation of everything expressing it [3]. That means it is the holistic composition of the technical domain model (including the constraints), code and documentation. Compared to a classic domain driven approach nothing has changed in this respect. Only – and this is the added value – the model and the language to describe the model are made much more explicit! They evolve as part of the whole and are always up to date.

### III. EXPERIMENTS

#### Implementation of Satellite Domain Model with Layered Architecture:

Let us discuss the real time satellite domain implementation in brief, the below architecture is on complete flow of the application with domain objects. This flow is based on the segregation of each layer with the components and technologies - Layered Architecture. For the DDD it is very important since we explicitly separate Domain Level [7], which lives between our Infrastructure and Application layers, like on the below figure. Note, that isolating of your Domain Level is very important.

![Satellite Application Architecture Diagram](image)

**User Interface (Presentation Layer)**

Responsible for presenting information to the user and interpreting user commands. The User Interface client is an Eclipse RCP based application composed of the GUI component and GUI Service component:

**GUI Service Layer**

This is a thin layer which coordinates the application activity. It does not contain business logic. It does not hold the state of the business objects, but it can hold the state of an application task progress. The GUI Service component layer between the UI and other subsystems, implements the patterns to load data from the server as required, populate and maintain the Domain Session and hides all concerns related to
maintaining a consistent view of local changes “merged” with data from the server. It uses the same EMF domain objects as the rest of the system.

**Domain GUI Service Layer**

Each client maintains a “Domain Session” of data that has been retrieved from the Business Logic Server and is held locally. The users create, update and delete data in the Domain Session and then can choose to commit their changes to the server or discard them. The user can also refresh their Domain Session from the Business Logic Server to obtain changes that have been made by other users of the system or that have arrived via the service bus. Domain Services component are tightly coupled to the Domain Model it provides a Domain Session with undo/redo command stack and Domain Services to query and modify Domain Objects in the Domain Session.

**Domain Model**

This layer contains information about the domain. This is the heart of the business software. The state of business objects is held here. Persistence of the business objects and possibly their state is delegated to the infrastructure layer. The Domain Model subsystem is composed of the Domain Classes component and the Domain Services component:

**Business Service Layer:**

The Business Service Layer is a fundamentally multi-user component and each instance is able to support multiple concurrent requests from multiple Satellite clients and the service bus. Implements the non domain model business logic of the Satellite system and includes the business rules which are applied (both fixed programmatic rules and configurable rules applied using Drools). It is composed of one sub-component per business component. This layer includes the DAOs (Data Access Objects), which perform the actual database access / persistence.

**Infrastructure Layer**

This layer acts as a supporting library for all the other layers. It provides communication between layers, implements persistence for business objects, contains supporting libraries for the user interface layer, etc.

**Domain Aggregates**

An Aggregate is a group of associated objects which are considered as one unit with regard to data changes. The Aggregate is demarcated by a boundary which separates the objects inside from those outside. Each Aggregate has one Root. The Root is an Entity, and it is the only object accessible from outside.

**Domain Objects component**

The domain model is generated using the EMF domain model (data) classes used by all the other subsystems/components. Instances of these classes represent the data used by / flowing through the satellite system.

The same domain objects are used server side and client side. Server side, Teneo adds support for EMF domain objects to be used directly by Hibernate and generates the required hibernate mappings (mappings are modified as required using EMF annotations)[5].

**Satellite System Behavior**

A behavioral requirement describes how a user will interact with a system (user interface issues), how someone will use a system (usage), or how a system fulfills a business function (business rules). These are often referred to as functional requirements.

**Satellite Non-behavioral**

A non-behavioral requirement describes a technical feature of a system, features typically pertaining to availability, security, performance, interoperability, dependability, and reliability.

IV. RESULTS

*The GMF (Graphical Modeling Framework)*

Process

See more at http://www.eclipse.org/modeling/gmp/
**GMF – Result**

![GMF Diagram]

**EMF: Ecore Components**

The Ecore components are related according to this hierarchy [9]:

![Ecore Component Hierarchy Diagram]

**Generating Java code using the ecore and genmodel files in EMF:**

First and foremost we have to create two model files, the .ecore and the .genmodel model. Based on these two model files we can generate Java code by right-click on the root node of the .genmodel file as shown in the below figure and select Generate Model Code.

![Generate Model Code]

This will create the Java implementation of the EMF model in the current project, the generated code will consists of the following:

- model -- Interfaces and the Factory to create the Java classes
- model.impl -- Concrete implementation of the interfaces defined in model
- model.util -- The Adapter Factory

The central factory has methods for creating all defined objects via createObjectName() methods, for each attribute the generated interface and its implementation contains getter and (if allowed in the model definition) setter methods. Each setter has also a generated notification to observers of the model.

Each generated interface extends theEObject interface. EObject is the base of every EMF class and is the EMF equivalent of java.lang.Object. EObject and its corresponding implementation classEObjectImpl provide a lightweight base class that lets the generated interfaces and classes participate in the EMF notification and persistence frameworks. Every generated method is tagged with @generated. If you want to manually adjust the method you want to prevent that EMF overwrites the method during the next generation run you have to remove this tag.

**Sequence Diagram**

The data managed via different wire has been captured in the below predicted sequence diagram, here each layer plays vital role in order to perform effective performance and to avoid excess amount of making call to Business layer.

![Sequence Diagram]

Here the GUI client maintains a separate Domain Session per scenario - i.e. if a user is making changes to two scenarios, the changes for each scenario will be in
their own Domain Session and can be committed (or discarded) separately.

Domain Services are tightly coupled to Domain Session and are provided to query the data in the domain session, to navigate between domain aggregates in the domain session and to modify the data.

Domain Services also provide domain logic that is tightly coupled with the Domain Model component but cannot easily be implemented directly in the domain objects themselves, such as calculations involving multiple aggregates.

Domain Services are used client side (though there is no inherent restriction to prevent them being used server side).

V. ACKNOWLEDGMENTS

I am very grateful to my institution for their constant support. I sincerely thank Dr. N.N.S.S.R.K Prasad for his valuable guidance, suggestions, useful remarks, and productive discussion which helped me to improve the content and presentation of this paper.

VI. CONCLUSION

This paper has discussed the current trends and practical usage of domain model in software development, in particular the definition and use of domain-specific languages, and their advantages and disadvantages. It has been able to capture the future requirements and enhancements on domain specific languages. We have seen, how, where and when to integrate the model technologies which has been captured and presented with simplified manner by differentiating each layer. We strongly believe that the industry will increasingly adopt model driven architecture and technologies with adopting the modeling technologies and tools. It represents that UML as an important language act as a backbone for model driven architecture using agile process, because it act as a service bus between the developers, analyst and domain experts. Today we are able to solve many complex applications, projects in customer language, making them to understand in their own language it is through our Modern Driven and Domain specific tools. We still need to focus more on this area in order to make more enlightening to reach to developer, analyst and domain experts to make extensive use of MDA and DMS.

VII. REFERENCES

[1] [Evans2003] "Domain-Driven Design: Tackling Complexity in Software".