Applying Model Driven Engineering For Developing Enterprise Software Applications

Ajesh Appukuttan
Research and Development
Phi 21 Technologies Pvt Ltd
Bangalore, India
ajesh.appukuttan@phi21.in

Aravinda Ramakrishna
Research and Development
Phi 21 Technologies Pvt Ltd
Bangalore, India
aravinda.ramakrishna@phi21.in

Sunil Madhavan
Research and Development
Phi 21 Technologies Pvt Ltd
Bangalore, India
sunil.madhavan@phi21.in

***Abstract* – Scale and size of enterprise software development has been increasing tremendously in the last couple of years. To handle this complexity, different architectural styles have been proposed and along with that a lot of frameworks supporting them have become available. The primary purpose of all these developments is to reduce complexity and make it easier to build large scale enterprise applications. In our experience such developments have been of value, but if software development remains labor intensive, such advances cannot be harnessed to their full potential. The primary challenge is the lack of uniform understanding of capabilities as well as best practices of these frameworks and architectural styles by everyone involved. Another challenge is the ability to handle with agility the change that is either based on changing business needs or technology. The ability to automate the development of software using Model Driven Engineering techniques can greatly address the challenges mentioned previously. This would involve generation of executable code as well as other artifacts such as database tables, configuration files etc. from formal models and ensure that the right patterns and best practices are followed in the generated artifacts. Handling changes based on business or technological needs can also be easily managed.**

Keywords— Enterprise Software Development, J2EE, Domain Driven Design, Microservices, DSL, REST.

I. INTRODUCTION

Companies big and small across the world and across different domains are in a race to utilize software and technology to create more value. This has led to an increase in complexity in the development of enterprise software. Not only do they automate increasingly complex business processes but also must integrate with a multitude of other systems. Adding to all this complexity are the advances in technology which is bringing in newer and better ways of doing things. To take advantage of these advances, software already developed needs to undergo constant change. Another dimension driving change in existing software is the fast-changing business needs. To top it all, software development is a labour-intensive activity and the gap between demand and supply of experienced professionals skilled in multiple technologies is rapidly increasing.

Comparing the enterprise software built a couple of years ago with the software being developed today will provide a better perspective. Earlier enterprise software was typically a web application with a browser-based user interface having server-side business logic storing data in a database typically used within the corporate mostly by employees. Today's applications tend to be multiple cooperating distributed services which can be accessed either through the browser, mobile or Application Programming Interfaces (API) which is also available to the external world. This brings in a set of whole new challenges around security, scalability, and performance. There are a set of patterns as well as frameworks implementing these patterns to address these challenges, but this involves following the right set of guidelines and best practices. Typically, the best practices and guidelines are formally enforced using reviews which not only take up bandwidth of experts but also are prone to human error. Generating artifacts using Model Driven Engineering (MDE) can not only ensure things are done the right way but also save a lot of human time and effort and help deliver more in a shorter time period. Introducing changes to existing software is easier as large-scale changes in generated artifacts can be done quickly by enhancing the code generator and thereby eliminating human effort in making those changes.

II. OUR EXPERIENCE WITH MDE

Our 20+ years of experience is in developing enterprise software products primarily on Java based technologies. We got introduced to MDE in 2004 while developing enterprise software products in the Telecom Expense Management Domain.

This product was developed as an n-tier J2EE [1] web application using standard J2EE components like EJB's (stateless session beans), Hibernate, Struts etc. We used Unified Modelling Language (UML) [2] to represent the model of our application. The models were in the form of class diagrams and used only the structural modelling capabilities of UML. This model was further embellished with stereotypes and tagged values to provide guidance to the code generation engine. We used an open-source code generation engine AndroMDA [3] and plugged in our custom code generation templates to generate multiple artifacts like Hibernate POJO classes, Session Beans, database schema as well as multiple XML configuration files. The primary objective of using MDE was to accelerate the development through automation as well as ensuring that guidelines and best practices were followed by enforcing them through code generation. Expert architects and engineers were responsible for ensuring that code generation templates follow best practices and guidelines. This ensured that uniformity was maintained in the quality of software being developed irrespective of the level of expertise.

Starting from year 2014, we were responsible for developing another large enterprise business software which had very tight deadlines and involved a large team of developers (100+). We had to build the development team by recruiting new members and most of them were not very familiar with the technologies used. The usage of MDE in this project was of great help to accelerate the development while ensuring that guidelines and patterns mandated by the architecture team were followed by embedding it in the code generator. Given the size and pace at which the team was growing, it would have been impossible to ensure uniformity in the absence of MDE. The code generation technology used was similar to what we had used previously.

Microservices [4] have been gaining a lot of ground along with Domain Driven Design (DDD) [5] becoming more

mainstream in the past couple of years. DDD advocates a model-based approach to developing software. These models represent the reality of the business domain which are represented in the form of "Aggregates" [6], "Entities" [7], "Value Objects" [8], "Commands" and "Events" [9]. Supporting these concepts are Application and Domains Services [10], Repositories [11] which provide API's as well as data persistence capabilities to support the domain layer. A Domain Specific Language (DSL) [12] to model the software using modelling elements provided by DDD can go a long way in combining DDD with MDE.

We have been exploring ways to use MDD to generate code from DDD models. Our initial thoughts were to use UML by extending it using appropriate stereotypes and tagged values to build DDD models. We realized very soon that we needed to look at an alternative mechanism as we did not find UML and DDD a natural fit. This is when the idea of developing a DSL which provides a natural representation of DDD concepts was considered. We came across "Context Mapper" [13] which provides DSL for building DDD models. Context Mapper DSL is developed using "XText" [14] which provides a solid framework for developing domain specific languages and excellent integration with Eclipse Integrated Development Environment (IDE) [15] providing support for syntax highlighting, syntax checking autocomplete etc., which is typically expected from an IDE. XText also provides the capability to generate a parser based on the grammar for the DSL, which can be used to build an object model. This object model can be traversed and integrated with code generation engine to generate various artifacts.

We have utilized the modelling capabilities of "Context Mapper" as well as the underlying parsing capabilities to read the model as input for the code generation framework that we built on top of the capabilities provided by Context Mapper. With this approach, we were able to overcome the challenges faced while using UML for specifying DDD models.

III. BENEFITS OF USING MDE

The key benefits that we derived by using MDE are listed below.

- Accelerate the pace of software development through automated code generation and increased productivity.
- Guaranteed compliance with guidelines and best practices as these are embedded into the code generator.
- Better application quality by eliminating human error and greater uniformity. Knowledge and experience of experts when embedded into generated code ensures that all team members can take advantage of it. This also ensures that this knowledge and expertise is not lost due to attrition.
- Relatively less time spent on reviews as the generated code need not go through code review process.
- More time spent on formal modeling and since the code is generated from the model, they both are always in sync. This is unlike typical cases where code and model representing the design diverge with time.
- Large scale changes in generated code can be done rapidly. One such instance is when we had to support a new database management system for the product we were developing. The entire database schema (consisting of 250+ tables etc.) was generated from the model and it was a matter of adding a new generator to generate the schema for the new database management system.
- Embrace continuous improvement by improving and enhancing the generator on a continuing basis.

IV. MDE CHALLANGES

The challenges that we faced while implementing MDE are elaborated below.

- UML, which was the standard for building software models till a couple of years back, was not found to be a natural fit with the advent of approaches like DDD. These challenges were resolved to some extent with the advent of DSL's.
- DSL provides a very solid foundation for MDE, but the challenge lies in defining the language and building surrounding infrastructure. In the recent past this has been addressed by availability of tools and frameworks for building new languages and their integration with widely used IDE's.
- Even though we have made advances and overcome challenges as mentioned previously, MDE is not mainstream yet and it has still not found widespread acceptance in the software industry.
- Expertise on MDE is not easily available in the software industry and hence there are not enough people to evangelize the MDE approach in the software industry. Most of the time MDE endeavors fail due to this lack of expertise. We also find that there are not many books, tutorials etc. available as compared to other areas of the software industry.
- Misconceptions and doubts about MDE - One common doubt the industry has in general is - "Will generated code be as efficient as handwritten". Fears about emphasis on modeling making software development process heavy which is a legacy of late 1990's.

V. GENESIS A CASE STUDY

Genesis is a model-based code generation framework that we have developed for our internal use. In the year 2020 a couple of us started a company to develop enterprise software products in the governance domain. As a very small team, we were looking at ways to automate instead of trying to add more members to the team to meet the time to market schedules.

The product that we were building was based on Microservices Architecture and we intended to use DDD for creating our models. Since we wanted the modeling language to be a natural fit for DDD we started exploring ways of developing a DSL that would suit our needs. We realized that building a DSL would not only be a complex project but also require a lot of expertise in this area. The effort to develop and maintain the DSL will also be substantial. Fortunately, we were able to find an open source DDD modeling framework Context Mapper which had all the capabilities that we were looking for, like support for DDD concepts, ability to parse and build an in-memory object representation of the model which can be used by code generation engine, IDE support for creating the models.

We developed a code generation engine that can read models created using Context Mapper. The key component of this code generator is the driver which reads the model and creates an in-memory object representation. Depending on the technologies used in the project, the appropriate code generation modules can be plugged into the driver, thus providing an extensible framework. The modules contain the logic and the templates to generate the code. The code generators support generation of Entities, Value Objects, Command and Event classes, Application and Domain Services as well as REST [16] endpoints. Code that needs manual modification is generated once and handwritten logic can be added after it is generated for the first time.

A pictorial representation of the same is shown in Fig. 1. As seen in the figure, the domain model is read by Genesis and depending on the configured modules the appropriate code is generated. For example, currently we support persistence to either relational database or document/graph database. Based on the configured module, the generated entities will be suitably annotated. Hints can also be provided as part of the model to further control the way code gets generated.

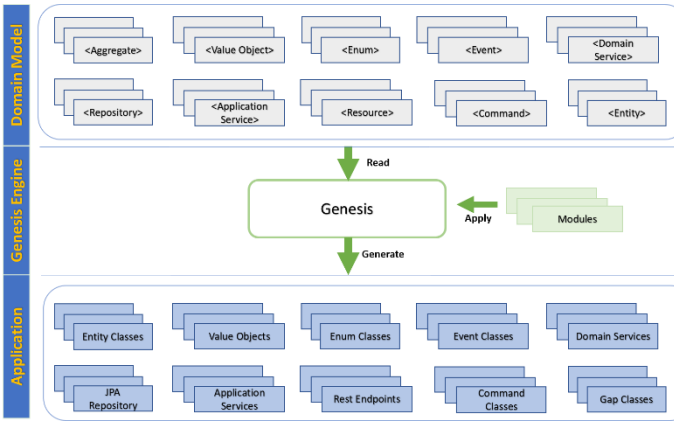

Fig. 1. Genesis Framework

A snippet of one of our domain models is shown in Fig. 2. As seen in Fig. 2, our model is defined using DSL in plain text format. This model will be used as input for the generator.

```
Aggregate Policy {

    CommandEvent AddPolicy {

        hint="init-true"
        String name required
        String description required
      ...
    Entity Policy {
        hint = "type=ARANGO_DOCUMENT"

        aggregateRoot

        private String id key
        String name required
        String description required
        String owner
              ...

    Flow Policy {
        command AddPolicy delegates to Policy emits event PolicyAdded
        command ChangePolicyDescription delegates to Policy emits event PolicyDescriptionChanged
        command ChangePolicyOwner delegates to Policy emits event PolicyOwnerChanged
        command ChangePolicyStatus delegates to Policy emits event PolicyStatusChanged
        command ChangePolicyOwnerEmail delegates to Policy emits event PolicyOwnerEmailChanged
        command ChangePolicyDetails delegates to Policy emits event PolicyDetailsChanged
            ...
Resource PolicyDefinitionRestService {
    inject @PolicyDefinitionService
    inject @PolicyDefinitionQueryService

    String add(@PolicyDefinitionDto dto) POST path = "/policyDefinition" => PolicyDefinitionService.add;
```

Fig. 2. Model Snippet

Fig. 3 has a snippet of code generated from the model shown above. It shows entity class with persistence related annotation for a document database, command, and event classes as well REST endpoint interface generated from the model. All the technology related code is being generated from the model. The handwritten code is limited to business logic.

```
package com.phi21.g21.metadata.policy.domain.mode

import org.springframework.data.annotation.Id;

@Document("POLICY")
public class Policy {
  ...

package com.phi21.g21.metadata.policy.domain.command;

import org.apache.commons.lang3.builder.HashCodeBuilder;

public class AddPolicy implements java.io.Serializable {
  ...
package com.phi21.g21.metadata.policy.domain.event;

import org.apache.commons.lang3.builder.HashCodeBuilder;

public class PolicyAdded implements java.io.Serializable {
  ...

package com.phi21.g21.metadata.policy.adapter.rest;

import org.springframework.web.bind.annotation.*;

@CrossOrigin(exposedHeaders = "*")
public interface PolicyRestService {
    @PostMapping(value = "/policy", consumes = { "application/*" }, produces = { "application/json" })
    default public ResponseEntity<String> add(
            @RequestBody com.phi21.g21.metadata.policy.adapter.rest.dto.PolicyDto dto) {
        throw new RuntimeException("Not Implemented.");
    }
```

Fig. 3. Code Snippet

VI. Conclusion

We have used Genesis framework to develop two modules of our governance product till now. Substantial benefits are seen in terms of better productivity and the quality of the artifacts. We can consistently generate approximately 60-65% of the code. The remaining 35-40% handwritten code includes business logic and automated unit test cases.

In conclusion, we believe that Model Driven Engineering will become an important part of mainstream software. The primary driver for this is the lower cost of building and maintaining software systems developed using MDE. Another important driver is the scarcity of skilled professionals leading to a need for more automation. The growing popularity of DSL's and availability of tools and frameworks to support them will also act as an enabler for the widespread adoption of MDE.

References

[1] Java 2 Platform Enterprise Edition Specification V1.4, JSR 151, 2004. [Online]. Available: https://jcp.org/en/jsr/detail?id=151.

[2] OMG Unified Modeling Language Specification, 1.5, Object Managment Group, March 2003. [Online]. Available: https://www.omg.org/spec/UML/1.5.

[3] AndroMDA. (3.5). [Online]. Available: http://www.andromda.org/.

[4] Sam Newman, Building Microservices. Sebastopol, CA, USA: O'Reilly Media Inc, 2021.

[5] Vaughn Vernon, Implementing Domain-Driven Design. Westford, MA, USA: Addison-Wesley Professional, 2013.

[6] Vaughn Vernon, "Aggregates, " in Implementing Domain-Driven Design. Westford, MA, USA: Addison-Wesley Professional, 2013.

[7] Vaughn Vernon, "Entities, " in Implementing Domain-Driven Design. Westford, MA, USA:Addison-Wesley Professional, 2013.

[8] Vaughn Vernon, "Value Objects, " in Implementing Domain-Driven Design. Westford, MA, USA:Addison-Wesley Professional, 2013.

[9] Vaughn Vernon, "Domain Events, " in Implementing Domain-Driven Design. Westford, MA, USA:Addison-Wesley Professional, 2013.

[10] Vaughn Vernon, "Services, " in Implementing Domain-Driven Design. Westford, MA, USA:Addison-Wesley Professional, 2013.

[11] Vaughn Vernon, "Repositories, " in Implementing Domain-Driven Design. Westford, MA, USA:Addison-Wesley Professional, 2013.

[12] Martin Fowler and Rebecca Parsons, Domain Specific Languages. Westford, MA, USA: Addison-Wesley Professional, 2010.

[13] Context mapper. (6.6). [Online]. Available: https://marketplace.eclipse.org/content/context-mapper/.

[14] Eclipse XText. (2.24.0.v20201130-1016S). [Online]. Available: https://github.com/eclipse/xtext.

[15] Eclipse. (2020-12). [Online]. Available: https://www.eclipse.org/.

[16] R. T. Fielding, "REST: Architectural Styles and the Design of Network-based Software Architectures," PhD dissertation, University of California, Irvine, 2000 [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.