The use of Model-Driven Architecture and Transformation in SPL

Malviya, Ankit — amalviy1@hawk.iit.edu
Sharma, Bijay — bsharma@hawk.iit.edu
Raychaudhury, Shaalmali — sraychau@hawk.iit.edu
Duggappa, Poojitha V — pduggappa@hawk.iit.edu
Maurya, Shailesh — smaurya@hawk.iit.edu

1. Introduction

Model-Driven Architecture and software Product lines (SPL) are the two different concepts which support reusability. With the help of SPL Engineering, development of software product and software-intensive systems can be done in a shorter time and at lower costs. Model Driven architecture is one of the concepts which is attracting software development researchers. MDA basically uses concepts of models at various levels using abstraction and model transformation from one phase of the software to another software. Model transformation is used also to transform one source model into another target model. This transformation increases the reusability in the software development process. Since both concepts facilitate intensive reuse strategy, so by combining them can bring significant improvement to software development. In the following sections, we will explore the use of Model-Driven Architecture and Transformation in SPL approach of software development.

1.1 Software Product Line

According to Software Engineering Institute (SEI), a *software product line* (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1].

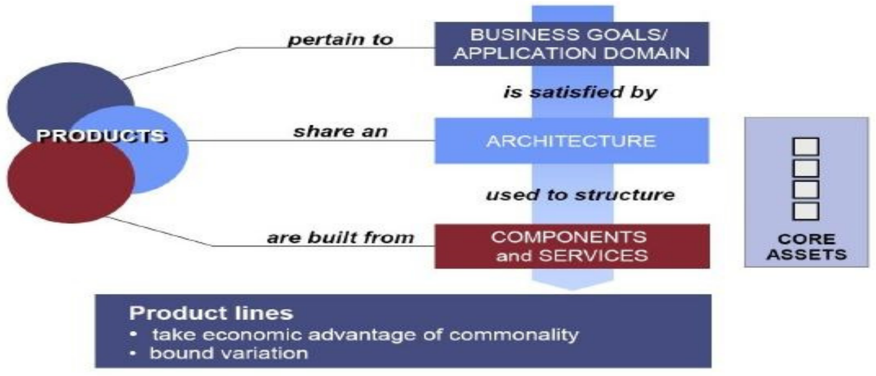

Figure1. Software Product Line

To overcome from resource shortage, organizations take help from "Product line". Various organizations of different types and sizes have suggested on implementing product lines properly and efficiently can produce many benefits and they have got a competitive edge. Example organizational benefits include:

- 10 times productivity improvement.
- 10 times quality improvement.
- 60% cost decreased
- 87% decrement labor.
- Time to market decreased by 98%
- Capable to come into new markets in months instead of years [1].

In other words, we take the advantage of commonalities between a set of related products which aims a specific market segment. The manufacturing industry has been utilizing Product lines from a long time and thus increasing the productivity by taking the advantage of commonalities among different available products. These products which already exist and are in operation from a log time are legacy systems. Boeing, Ford, Ericsson and McDonalds are few companies which apply product lines.

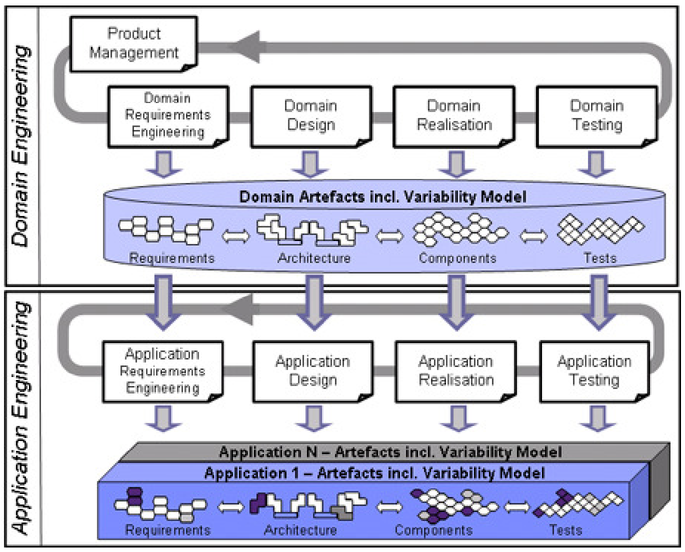

Figure2. Software Product Line framework

1.2 Model-Driven Architecture

Model-Driven architecture has been created by Object Management Group (OMG) initiative. This is an approach to the development of systems using models and mapping of models for implementations. MDA consists of a set of rules and guidelines from the structuring of specifications. There are different models are present. Platform-Independent model is the basic one. It can be automatically mapped to platform specific models. One platform independent model can be mapped to one or more than PSM. MDA is like a domain engineering which supports architecture based on models.

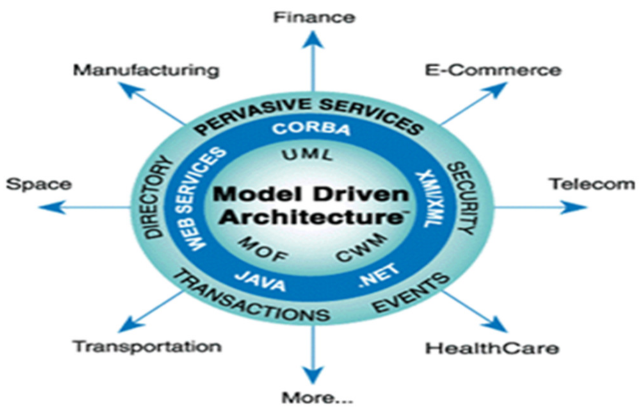

Figure3. OMG's Model-Driven Architecture

Figure 3 shows Model Driven Architecture. The MDA is independent of language, vendor and other things like middleware. The center of the architecture consists of modeling standards. These standards are UML, the MOF, and CWM. We first make a platform independent application model in UML. This is only an appropriate core model. Now, the platform specialists come into the picture and will convert this general model to a platform dependent model according to a particular platform. These platforms can be CCM, EJB or MTS. If the mapping is done in a standard way, then automatic conversion can be carried out. The thin ring surrounding the core covers the target platforms. Business and technical run-time semantics of the applications are handled with the help of these platform specific models. Once, we're done with platform specific model, we can go ahead and create code. The quality of semantics included and run-time behavior depends on the completeness of the platform specific UML models.

The flagship concept of MDA:

<u>Model</u> - Models consist of sets of elements that describe some physical, abstract, or hypothetical reality. Goods model can be used to relate different part of the implementation of a system. We can say they are a mean of communication. They are economical to build than to build a real thing. We can use models to make a rough idea to a detailed blueprint of a system to a fully executable model.

<u>Meta-model</u> - Like meta data, meta model is a model of a modeling language. Metamodel specifies the structure, semantics and constraints for a group of models. These groups share a common syntax and semantics. A metamodel captures can capture a model.

<u>Platform</u> - It is basically execution environment specification. Platform examples are Linux, Solaris, and windows.

1.3 Model Transformation

One of the notable features and the rising research field of MDA is Transformation. There are three major types of modes in MDA, namely - *Computational Independent Model* (CIM) which involves system functionality's specifications, *Platform Independent Model* (PIM) and *Platform Dependent Model* (PSM).

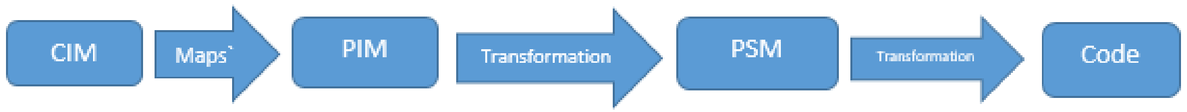

Figure4. Model Transformation

Our major focus is on PIM and how it is automatically mapped to a PSM. The key point is here that a PIM can be mapped to PSM. Means we can generate different platform specific model from a single platform dependent The CIM is mapped to PIM. The PIM is transformed using transformation rules to PSM and from PSM, code can be generated. The more effectively a model is generated less manual work is done.

We can also classify transformations into two different ways, namely:

- Model(PIM) to model transformation(PSM), and
- Model(PSM) to code transformation.

Model to model transformation also includes mapping of CIM to PIM. Following are common techniques - *Common Warehouse Meta-Model Transformation* (CWM), *Graph transformation* (GT), *Extensible Style Sheet Language Transformation* (XSLT), *Query, View and Transformation Approach* (QVT) etc. To generate code from PSM, we can use following techniques - *Visitor based approach* and *Template based approach*.

2. Discussion of relevant topics

This section will discuss certain software development approaches along with an Industrial Case Study at Ericsson, which will illustrate the use of Model-Driven Architecture and Model Transformation in Software Product Lines.

2.1 Modden

The consideration of the reuse of software product line should start from conception phase of software development artifacts. After implementation of the system, these artifacts can be stored in a library for future use by other domain application. This is a way of reuse. In model-driven paradigm, an application can be thought of a set of transformation which ends up to a final system. The important point to note here is during the transformation, generally, we do not consider reuse techniques. One of the SPL approaches that use model-driven techniques is *Modden*. We can enhance model driven techniques by integrating these two methods and by utilizing these artifacts in different phases of software development process. This can be done in two processes. The first is to develop core assets. The second is to develop specifies product line assets like UML profile and a model development supporting tool. An overview of Modden approach is shown in figure 5. The Modden approach composed of two processes namely, *MDA-DE* and *MDA-APP*. Software and System Process Engineering Metamodel (SPEM) v2.0 concepts and terminologies are required to give the specification of these process.

MDA-DE: It is a Domain Engineering process. It is designed to develop reusable core assets. There are four different phases of it. These phases are *CIM-DE*, *PIM-DE*, *PSM-DE* and *Code*. These phases produce artifacts which are kept in Asset repository. The main purpose of this storage is reuse.

MDA-APP: It is an Application Engineering process. It develops a specific asset for the product line. Like MDA-DE, it has also four phases *CIM-APP*, *PIM-APP*, *PSM-APP* and *Code*. Artifacts generated by these phases are also stored in a repository for reuse.

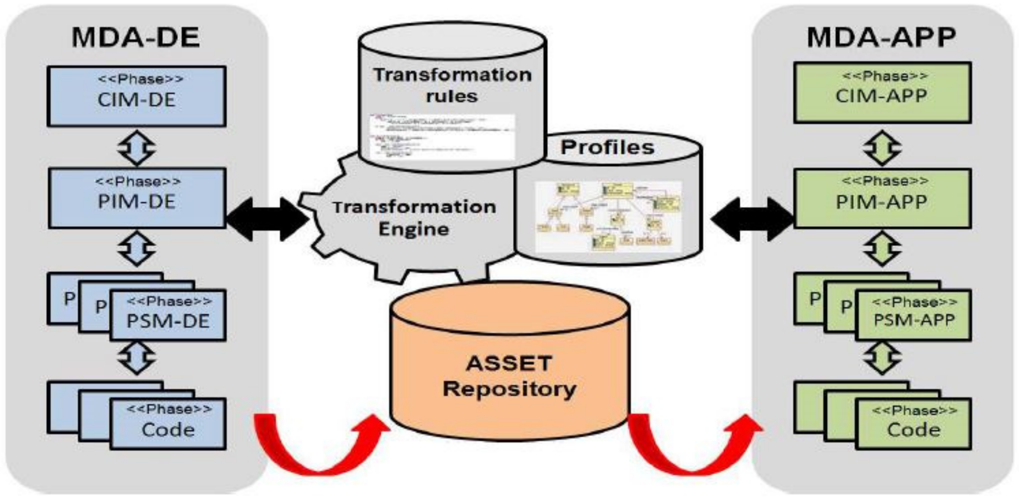

Figure5. Overview of Modden

Two things that support the process are profiles and transformation rules. Specific knowledge of variability and commonalities in feature diagrams are specified by profiles. So, profiles are necessary. Our profile encapsulates feature concepts and a rule. The rule is used to create a profile as per the feature model. Traceability between models is preserved using these profiles. So, profiles are also used in the maintenance of software system. Support to these process is provided by:

- The Modden tool
- UML editors
- tools executing transformation rules.

2.1.1 MDA-DE Process

A SPEM/MDA is shown in figure 6. It partially depicts MDA-DE process. Here the method is a library. The method describes reusable definitions such as *Disciplines*, *Tasks*, *Work Products*, and *Roles*. These reusable definitions will be used during the lifecycle of the process. The skill and responsibilities of an individual or a group are identified by a Role. Individual performs tasks during the execution of the process. Tasks can be grouped into the major area. These major areas are called Disciplines. They may consume/produce work products. We use Asset repository to store work products. These work products are used for reuse and for every domain engineering. There are four kinds of such artifacts.

Model: It is produced by a process role. It can be also produced by a transformation in the process execution.
Transformation: They consists of rules needed for model transformation. They also contain rules for code generation in the process execution.

<u>Extra Model</u>: It is useful for the purpose of documentation.
<u>Profile</u>: A UML profile is identified by a profile to base the modeling.

UML models are transformed automatically using Transformation rules in Model-driven Architecture. We can get one or more target models from one source model.

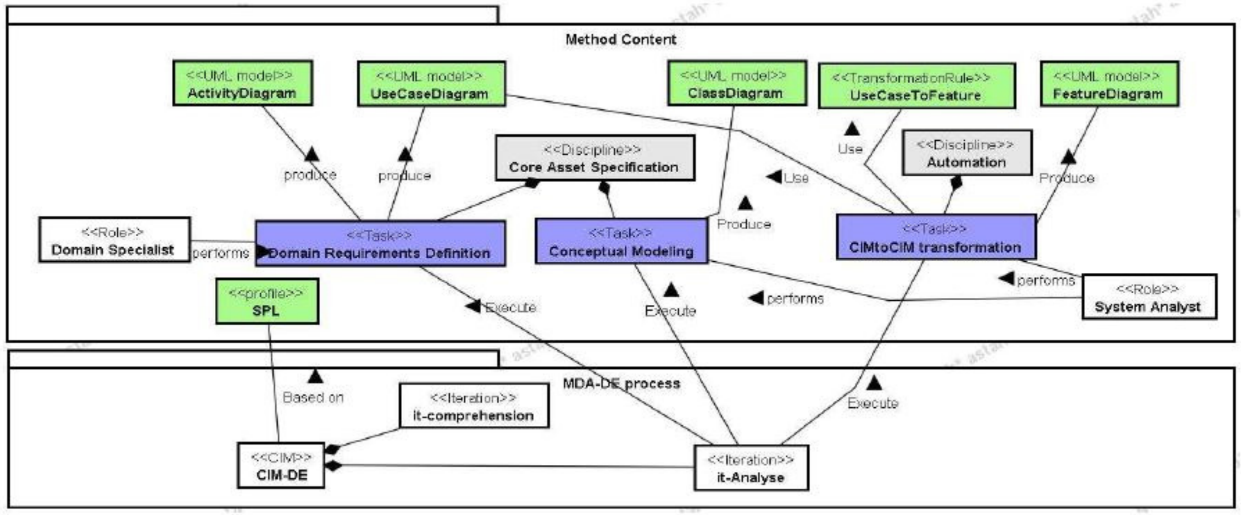
Method Content
<<UML model>> ActivityDiagram
<<UML model>> UseCaseDiagram
<<UML model>> ClassDiagram
<<TransformationRule>> UseCaseToFeature
<<UML model>> FeatureDiagram
<<Discipline>> Core Asset Specification
<<Discipline>> Automation
<<Role>> Domain Specialist
<<Task>> Domain Requirements Definition
<<Task>> Conceptual Modeling
<<Task>> CIMtoCIM transformation
<<Role>> System Analyst
<<profile>> SPL
produce
produce
Produce
Use
Use
Produce
performs
performs
performs
Execute
Execute
Execute
MDA-DE process
Based on
<<Iteration>> it-comprehension
<<CIM>> CIM-DE
<<Iteration>> it-Analyse

Figure6. A fragment of MDA-DE specification

The life cycle of a process has a set of sequential phases. These phases are performed iteratively. For model driven architecture, we have modeling of CIM, PIM and PSM and codification. We can use <CIM>, <PIM>, <PSM> and <Code> to represent these phases. We can relate each phase to a UML profile. These UML profiles are actually defined to depict the specifications of a particular domain or platform. Thus, MDA-DE process can be of two types static and Dynamic dimensions and as per SPEM terminology, we can call them method content and process respectively. For the MDA-DE method content ten disciplines were defined namely - *domain scope*, *asset search*, *core asset specification*, *domain analysis and design*, *domain commonalities and variability specification*, *variability management*, *asset implementation*, *asset cataloging*, *validation and verification*, and *automation*. Roles are associated to individuals when accomplishing activities. In MDA-DE process six roles are defined:

- domain specialist, individuals who are familiar with the domain concepts
- users, who know the domain through the use of existing systems
- system analyst, who is responsible for technical definitions during domain analyses
- architect, who is in charge of the domain design
- product developer, who is responsible for the codification
- tester, who is in charge of assets testing.

For each task of each discipline, it indicates the input and output work products. Steps which are required to be performed to complete the task are also specified, guiding the user during the process execution. For example, in the task Commonalities and variabilities identification, the input work product is the Use Case model. we have specified five steps for this:

- identify related use cases,
- create a feature on the feature model to group these use cases,
- identify the variabilities associated to this new feature,

- classify features according to the profile, and
- identify the relationship between the features.

The Dynamic dimension represents the process life cycle phases and their related iterations. The MDA-DE process comprises four phases:

- CIM-DE that models domain requirements,
- PIM-DE which is related to the domain design artifacts,
- PSM-DE which is related to the domain design in a specific platform, and
- Coding, where the asset implementation is fulfilled and kept for further reuse.

Multiple iterations can be in any of the phase and also each phase follows the discipline execution. As per the task, these disciplines can happen in one or more phases. A discipline specified in the method content can be found in different process phases. At the end of the iteration, work product produced by the MDA-DE process is kept in the Asset Repository for reuse during the implementation stages of other application.

2.2 Baseline-Oriented Modelling

Baseline-Oriented Modelling (BOM) is a framework which utilizes the model transformation and software product line techniques. The software applications produced by BOM follows PRISM architectural models. BOM follows basic concepts of Model Driven architecture by building domain models. These models can be converted into platform independent models. Once we get the PIM models, these can be compiled into an executable application which is a platform specific model. BOM is illustrated by the Expert systems.

As per Jackson, Peter in his book 'Introduction to Expert Systems', "an expert system is a computer system that emulates the decision-making ability of a human expert". The expert system understands the knowledge of experts and tries to utilize the reasoning process of such experts while solving problems in a certain domain. The architecture of such system is generic with components which are separate from each other and independent. Varying architectural elements, however, make these systems complex. Good thing is that MDA comes in support to over this complexity by bringing automation while software development. MDA uses models and platform independence technique in the software development process. This results in generating applications automatically. On the other hand, Software product line is used to overcome the variability problem. This also helps in controlling and minimizing the high costs of the software development process and to reduce the time to market of a family of products.

SPL basically uses a generic design shared by all the product family members. Baseline Oriented Modeling framework is based on this concept of generic design. BOM produces an Expert system as PRISMA architectural models based on SPL and by utilizing MDA approach. The PRISMA integrates two approaches. First is Component-Based Software Development and Second is Aspect-Oriented Software Development. The architectural elements through their aspects help to define the integration of both approaches. [2].

2.2.1 BOM: The Framework

We have emphasized on a diagnostic expert system to depict BOM. There are two types of variability management. The variants of the first variable are the domain feature and end user requirements. The variant of the second variability is application domain features. The application domain features determine the final product of SPL. The features of the first variability have been represented as variability points and their variants.

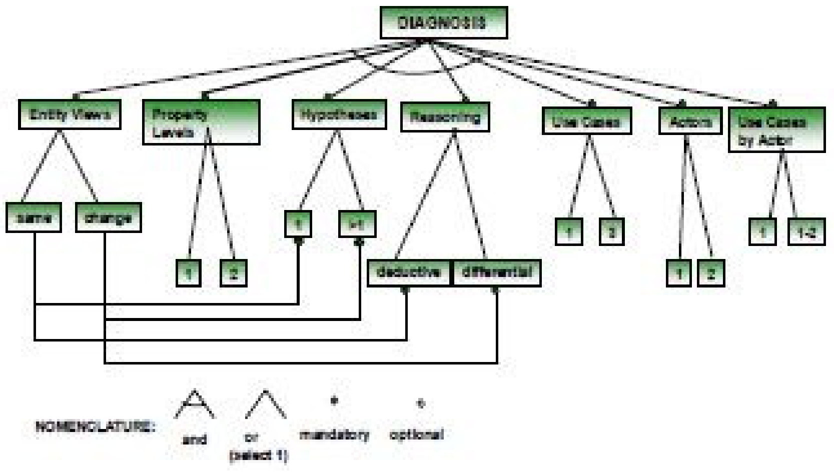

Figure7. Feature Model

The Feature Model is translated to an equivalent UML-class diagram with OCL constraints (see Figure 8): The Domain Conceptual Model. The variants are captured as instances of this Conceptual Model. The instances capturing the domain features in the medical diagnosis are: *entity view=same, property levels=2, hypotheses=14, reasoning=differential, use cases=3, actors=2, use cases by actor=2 by actor A & 1 by actor B*. The features of the second variability, i.e. variability of the application domain, are captured as instances of the Application Domain Conceptual Model (see Figure 9). We present an example of these instances in the medical diagnosis case study: *properties of level 0=cough, fever; properties of level 1=dry_cough, constant_fever; hypotheses of level 1=warth, parotitis; hypotheses of level 2=pneumonia, bronchitis; rules=IF (cough=true and fever=true and respiratory_dificulty=true) THEN syndrome=warth* [2].

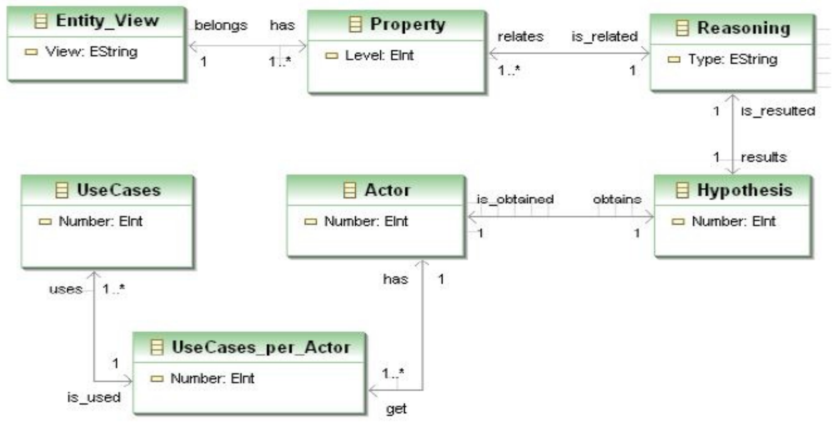

Figure 8. The domain conceptual model

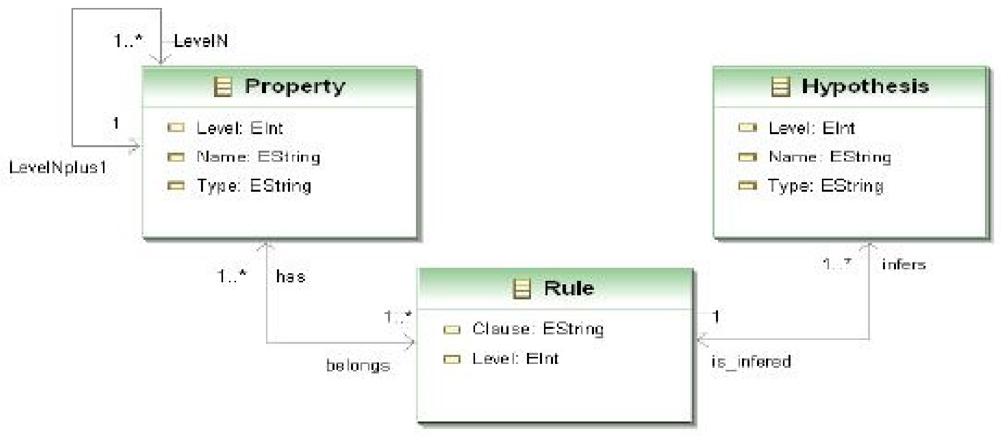

Figure 9. The application domain conceptual model

For this reason, there are two stages for modeling of the system:

<u>Stage 1</u>: First we make the Domain Conceptual Model which captures the variants of the domain in its instances (V1) and the model which shows how the system functions. We can then produce the corresponding base architectural model by using Query/Views/Transformations (QVT) OMG standard: QVT-Relations. Several base architectures can share a unique generic architecture.

<u>Stage 2</u>: Secondly we construct the Application Domain Conceptual model which captures the application variants in its instances. The instances of this model and the functional model of the systems are transformed into a PRISMA. an architectural model which is the final product of the SPL. This is the Expert System. The Expert system is in compliance with two variabilities and functionality selected. The second variability which involves the SPL of the application in a specified field i.e. SPL2 makes the base architectures to be decorated with the application domain features to produce the types of the PRISM software artifacts.

Further, the Domain conceptual model follows the V1 Metamodel (MM V1) and the application Domain Conceptual Model follows V2 Metamodel (MM V2). These meta-models are the UML 2.0.

2.2.2 The relations among metamodels

QVT transformations are utilized to establish relations among the metamodels. The relations are described by using the QVT-Relations language. The source and target metamodels are identified first. Then the correspondence among each element of the metamodels is then defined by taking into account the way in which their rules are represented via QVT. Here, the source metamodels are the V1 variability metamodel and the Modular metamodel. The target metamodel is the Skeleton metamodel. This means that the rules considered in the relations between the elements of the V1 variability and modular metamodels can only be of ‘check only type' and of ‘enforce type'.

2.2.3 The transformation among models

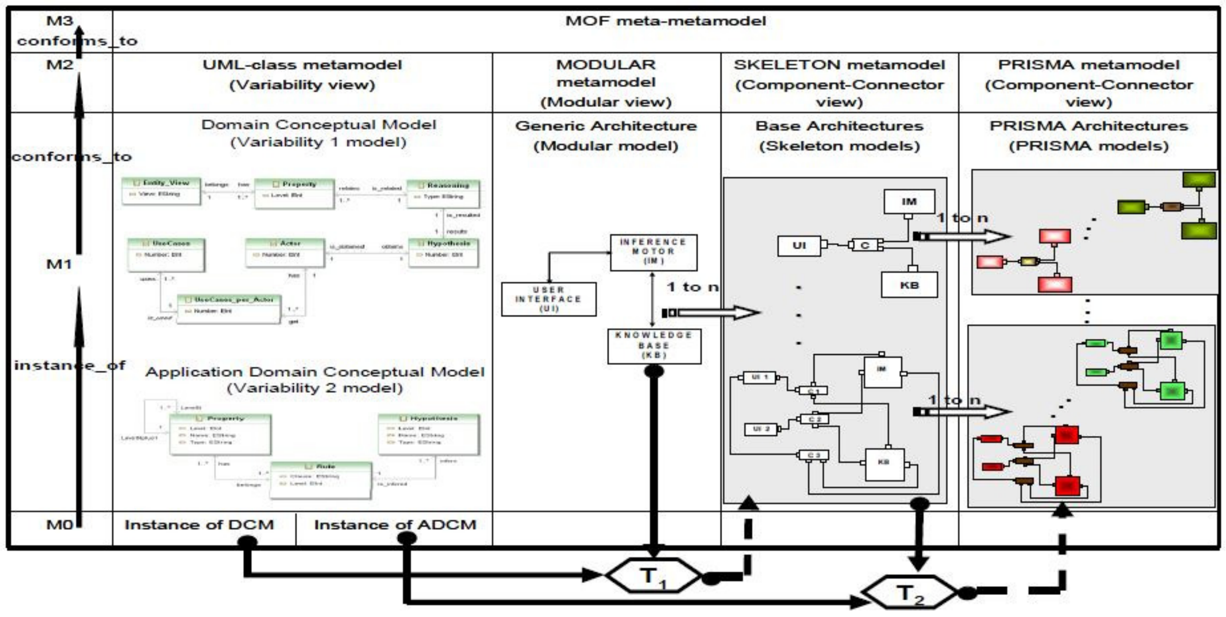

Figure 10. The model transformations T1 and T2 in BOM

Figure 10 depicts the model transformations are involved in the construction of SPL architectures. This figure describes how the transformation processes apply the model transformations performed at the model level. The modular model is made up of three modules: *InferenceMotor*, *KnowledgeBase*, and *UserInterface*. These modules are linked with each other via dependence relations. Then, the transformation task responsible for producing the skeleton model is performed for which the Domain Conceptual Model is used. A skeleton component is then produced for each module, and a skeleton connector is generated by each dependence relation. At level M1, the transformations T1 and T2 are carried out. A first skeleton model is obtained by the transformation T1, using the Domain Conceptual Model and the modular model as sources. This skeleton model is then refined by transformation T2 which uses the Application Domain Conceptual Model allowing the PRISMA model to be obtained as a refinement. In BOM, the model transformations T1 and T2 are executed by using QVT-Relations model transformations (these are PIM to PIM transformations). The QVT-Relations code (partial) is as follows:

```
transformation modules2components(mdomain:

mview, dcmdomain: dcm, ccdomain: ccview) {

//… Keys declaration

top relation ModulesModel2ComponentsModel

{ checkonly domain mdomain modulesModel:

mview::ModulesModel { };

checkonly domain dcmdomain varModel:

dcm::DomainConceptualModel { };
enforce domain ccdomain componentsModel:
ccview::CCModel {name=modulesModel.name};
}
where {
UseCase2Connector(modulesModel,varModel,
componentsModel); } }
//… Other relations
} [3]
```

In the end, PRISMA-MODELCOMPILER is the tool used to produce automatically executable code (in C#.NET) from PRISMA PIM generated model. PRISMA-NET middleware or PSM is used to execute the object code. The educational program diagnosis knowledge Base component in C# code has been shown below:

```
namespace KBE
{ [Serializable]
public class KBase: ComponentBase
{public class KBase string name:base(name)
{AddAspect (new FBaseE ( ) );
InPorts.Add("KnowPort","IDomainE","KNOW");
OutPorts.Add("KnowPort","IDomainE","KNOW");
} } } [3]
```

In BOM, the final user, which is application engineer, builds an Expert System (a product of SPL) by giving the features of the variabilities V2 and V1 as input using the instances of the Application Domain Conceptual Model and the Domain Conceptual Model respectively.

2.3 An Industry case study at Ericsson

This section describes an Industry Case Study done at Ericsson in Grimstad to show the integration of legacy systems with new systems and technologies. It also includes the possibility of model transformation in developing models that are platform and technology independent. Any system that is being operational and being used for a long time is a legacy system. We need to integrate these legacy systems in MDA in order to take its advantage. Ericson has developed two major telecommunication systems based on the multiple uses of existing platforms and development environment. We would understand a platform in our context, software artifacts being platform dependent or Independent. We would also develop a tool for model transformation.

2.3.1 Ericsson case

Ericson has developed two ways to deliver to provide GPRS solution to enable communication facility to mobile devices:

- Global System for Mobile communication (GSM)
- Wideband Code Division Multiple Access (WCDMA)

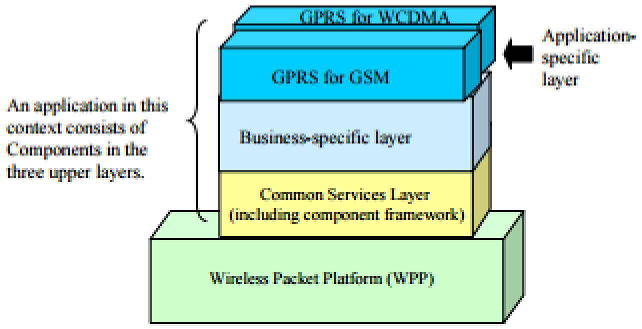
GPRS for WCDMA
GPRS for GSM
Application-specific layer
An application in this context consists of Components in the three upper layers.
Business-specific layer
Common Services Layer (including component framework)
Wireless Packet Platform (WPP)

Figure11. The GPRS Nodes software architecture

As we move during the development of a system from the requirement phase to executable phase, we need to use different diagram and models to represent them and relationship between different phases and MDA concepts.

<u>Use case model</u>: A use case is a list of actions or event steps, typically defining the interactions between a role (known in the Unified Modeling Language as an actor) and a system, to achieve a goal.

<u>Domain Object model</u>: A domain model is generally implemented as an object model within a layer that uses a lower-level layer for persistence and "publishes" an API to a higher-level layer to gain access to the data and behavior of the model.

These models are developed in requirement phase. These are further detailed to provide Design class diagram and sequence diagrams. These diagrams represent structure and behavior of the system respectively. In Model Driven architecture, a transformation happens from one complete model to a different model. A model can be a problem model or a solution model. We can present a problem domain with the help of UML. The problem domain model is platform independent model. The models which we develop in Analysis phase is also independent of platforms, so they are also PIM. The problem domain in the requirement phase is not computationally complete. When we come to design phase, it contains all the details from which a code can be generated. So, here it becomes computationally complete.

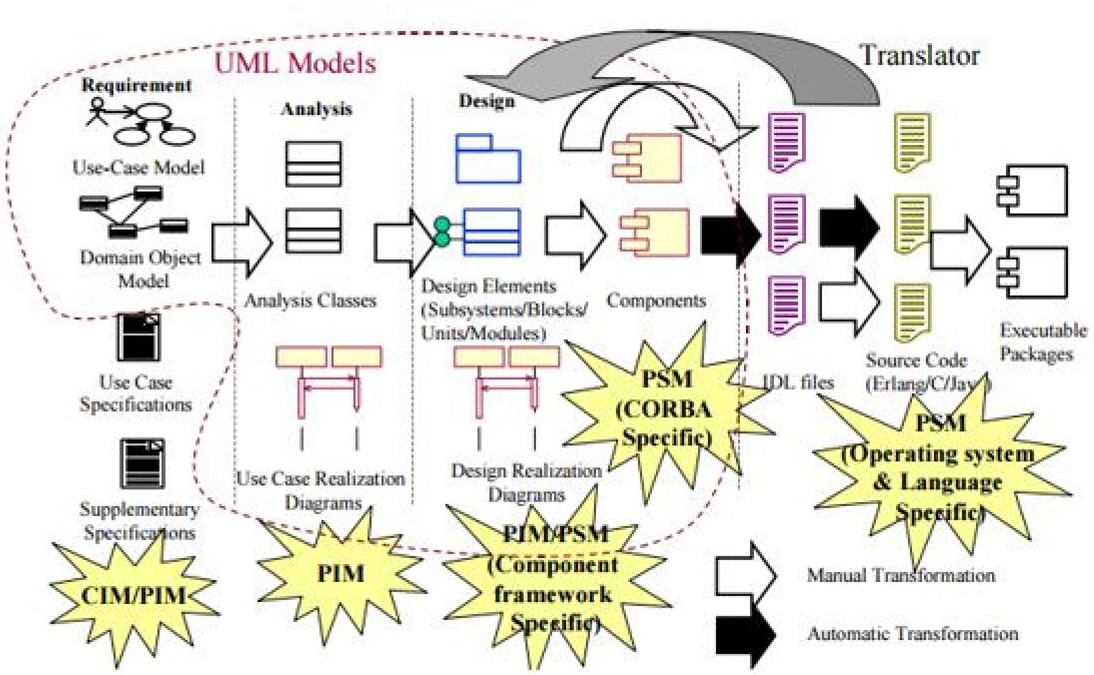
UML Models
Translator
Requirement
Use-Case Model
Domain Object Model
Use Case Specifications
Supplementary Specifications
Analysis
Analysis Classes
Use Case Realization Diagrams
Design
Design Elements (Subsystems/Blocks/ Units/Modules)
Design Realization Diagrams
Components
IDL files
Source Code (Erlang/C/Java)
Executable Packages
CIM/PIM
PIM
PIM/PSM (Component framework Specific)
PSM (CORBA Specific)
PSM (Operating system & Language Specific)
Manual Transformation
Automatic Transformation

Figure12. From requirements to executables

The design phase is platform independent model; however, it depends on the component framework. The component framework can be achieved in CORBA and OTP. A component framework is like a virtual machine which can be realized in CORBA and Erlang/OTP. The framework contains details and rules to be used by application developers. Developers use framework's services and template for programming. There are few concerns while doing transformations:

- Inspection and testing needs to be done to fix any inconsistency between -
  - written requirement and UML models,
  - UML model and the code, and
  - different UML models.
- Developers may update one model without updating another model.
- To save cost, not all the models are developed fully.

<u>The Translator</u>: The curved gray arrow in the above diagram is the Translator. The translator helps to do reverse engineering like process of creating PIM or PSM from code.

- Removing that parts of the code which is platform (Erlang/OTP & CORBA) specific like operation for starting, consistency check, transaction handling etc.
- IDL files are used to remove data types as Erlang being a dynamic type language, data type declaration is not required.
- UML support needed for model exchange.

The author has developed a new tool to support reverse engineering from code or design model sequence diagram.

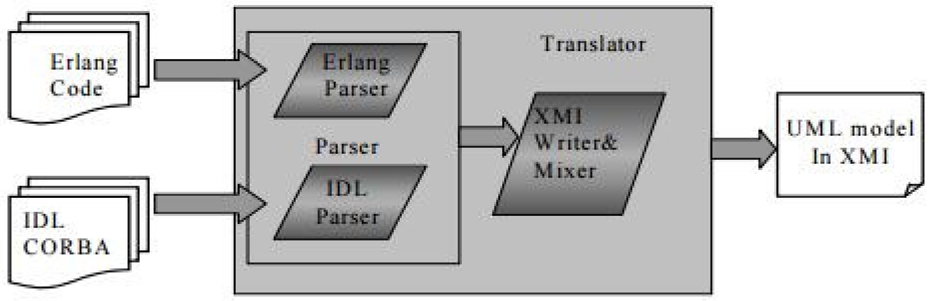

Figure13. The Erlang to XMI Translator

After the translation, the result is XMI. We need to parse a sub-part of the system as the parsing total system may take time. After getting different XMI files, these are merged using XMI mixer. The resultant model is dependent on the component framework but not on the COBRA, Erland, and OTP. It shows design model completely and does not contain any behavior specific information. One of the issue with Erlang which makes the transformation more complex is it allows using methods having the same name but different parameters in same software module and sometimes developers use this property to make the code backward compatible even though the guideline suggests otherwise. Each Erlang file is converted into UML class and the methods of Erlang files to a public method in UML class. The methods which were earlier dependent on OTP platform are removed to make the model platform independent. There is few advantage of creating an abstraction by transforming a PSM to another PSM.

- A change in the code is reflected in model. It means code and models are synchronic.
- We can utilize UML model in the development of systems in association with other models.
- Models are exchanged using XMI.

3. Conclusion

We have identified new approaches such as Modden and Baseline-Oriented Modelling which can produce Software Product Line utilizing the Model-Driven Architecture methodology and Model transformation techniques. We have also identified an Industrial case study done at Ericsson where they have integrated the legacy systems in MDA.

The Modden approach is based on SPL activities and support development through Model-Driven Architecture and Model transformations. In Modden, two software processes are defined: – MDA-DE (to develop the reusable core assets) and MDA-APP (to develop a specific product in Software Product Line). This term paper describes few details about the first process MDA-DE which evaluates the hypothesis that if SPL and MDA approaches are integrated it can enhance reuse during MDA processes transformation. Also, we described another such approach, namely, Baseline-Oriented Modelling (BOM). In BOM, we captured the features of Feature Model in conceptual models. These features are inserted, by means of QVT transformations, in the skeleton aspects of the architectural elements which make up the PRISMA models. Next, the Ericsson case study confirmed the possibility and low cost of developing a tool that helps to keep the UML model in sync with the code, thereby successfully integrating the SPL and MDA approaches along with the Model transformations.

4. References

[1] Software Engineering Institute (SEI), http://www.sei.cmu.edu/productlines/

[2] Modden: An Integrated Approach for Model Driven Development and Software Product Line Processes by Ana Patrícia Magalhães, José Maria N. David, Rita Suzana P. Maciel and Filipe Araujo da Silva

[3] Baseline-Oriented Modeling: an MDA approach based on Software Product Lines for the Expert Systems development by María Eugenia Cabello, Isidro Ramos, Abel Gómez and Rogelio Limón

[4] MDA and Integration of Legacy Systems: An Industrial Case Study by Parastoo Mohagheghi, Jan Pettersen Nytun, Selo and Warsun Najib