3 Variability Modeling

All difficult things have their origin in that which is easy, and great things in that which is small

(Lao-Tzu)

Contents

This chapter presents the analysis of the variability in the MeDIC information system. After the variation points and its variants are determined, a variability mechanism is applied to introduce and implement the variability. The variability mechanism applied for entity variability in the metric system is pattern mechanism [section 2.2.3.]. This chapter starts with explanation of all tasks that are executed in this master thesis.

3.1 Tasks

The goal of this master thesis was to find new meta-model design as the enhancement part of the MeDIC information system that addresses the variability issues. To achieve the objective described in the previous section, the following tasks were proposed:

Get to know about the topic through literature review Comprehend existing system about metrics and related work about variability. The purpose of this task is to get understanding about the current system and summarize basic theories about variability modeling that can be used to enhance the meta-model.

Investigate and analyze the requirements for the methodology The task starts with investigation the problem domain to find variation points in the current system. Analysis is focused in each variation point, as an enhancement study case in each meta-model to find a best solution pattern.

Development of a general modeling concept of the variability There are so

many variability concepts exist especially in Software Product Line (SPL). Based on study cases in the previous task, general modeling concept of the variability that applicable in the MeDIC system is developed. Summary about the general variability modeling will be present from this task.

Implementation of the variability in the MeDIC information system Describe the work plan for implementation with new meta-model. Graphic user interface will be proposed from several variation points.

Evaluation of the new meta-model Evaluating the new meta-model to define the deficiencies and the strength. Performing comparison between the current meta-model and the proposed meta-model, to evaluate the advantages of the works.

Documentation In parallel to all previous tasks described, the work must be documented in written form.

3.2 Variability in MeDIC system

3.2.1 Variability of Entity Specification

Variability in the MeDIC system focuses on varying degree of entity specification. The development of metrics is similar with paradigm of software development process, where the initiation phase is started with an abstract description of the problem domain. Several processes and inputs from user will develop the metrics from the abstract description into a model that can be implemented.

The variability term used in this work is probably different from the general concept of common variability that concern about product family with the variation of product component. Therefore, the following definition will be used throughout the work to guide the understanding of the work this master thesis.

> *"Entity variability is the varying degree of entity specification, start from the initiation form until the complete form of entity that can be implemented"*

The idea for modeling entity variability was started from the user requirement at initiation phase to propose a metric. Most of the users will propose a metric with general information that is written in textual description; mention about the general purpose of the metric, how the metric want to be visualized, and other requirements. With this rough textual description, parameters or attributes of a metric could not be determined explicitly. The description of metric later on will be expanded with specific attributes based on further input from user itself. The current meta-model of metric in MeDIC system is not support variation specification of the entity yet. The model only handles the first phase of entity

specification, which is textual description. Therefore, the functionality of metric could not be fully completed.

For example, the information needs specification of a metric. Information needs are the starting point to measure product or process. The current information needs are stored and represented as textual description. Whereas, in the development process based on GQM method (mentioned in section 2.1.2.), these information needs have to be formulated into questions. Furthermore, the questions will be used as references in proposing metric. Metrics that are created have to answer those questions. On the other case, information needs need to be categorized. Categorization of information needs will be used to realize reusability of metrics in organization. Before proposing a metric, user will be able to search the existing metric with similar information need based on the category. If the metric is found, that metric can be reused with some adjustment for their current project. Both of those functionalities haven't been supported yet in current MeDIC system.

Process for modeling metric entities variability will be divided into 2 phases:

Analysis Phases Based on the user requirements , variability analysis will start with defining all of variation points and its variants on the system. Every entity in metric meta-model will be analyzed to find the possibility of variability. The entity variability might occur from one or more sources listed below:

- An entity is defined over several process steps , an entity that is started with textual description and later on become more specific with attributes attached to that entity for implementation need, or
- An entity has variation evolution, possibility of an entity to be evolved into different type of entity, or
- An entity that has variant attributes or cardinality in different project.

Design Phases Model the entity variability with pattern as the variability mechanism. A set of predefined pattern will be determined first. Those patterns are design patterns that are frequently referred to as variability mechanism, provides a way to implement variability, such as 'Adapter', 'Strategy', 'Template Method', 'Factory', 'Abstract Factory', 'State', 'Builder', and 'Decorator'. Each of variation point will be model with the suitable pattern, chosen from that set of predefined pattern. Next step in this phase is design prototyping. Prototyping will provide a first look of actual implementation of the system. This prototyping will be discussed in detail in Chapter 4.

3.2.2 Analyze Variation Points in the MeDIC system

Before defining variation points in metric meta-model, the core aspect of metric meta-model will be introduced. The current metric meta-model of MeDIC system is shown in figure 3.1 below:

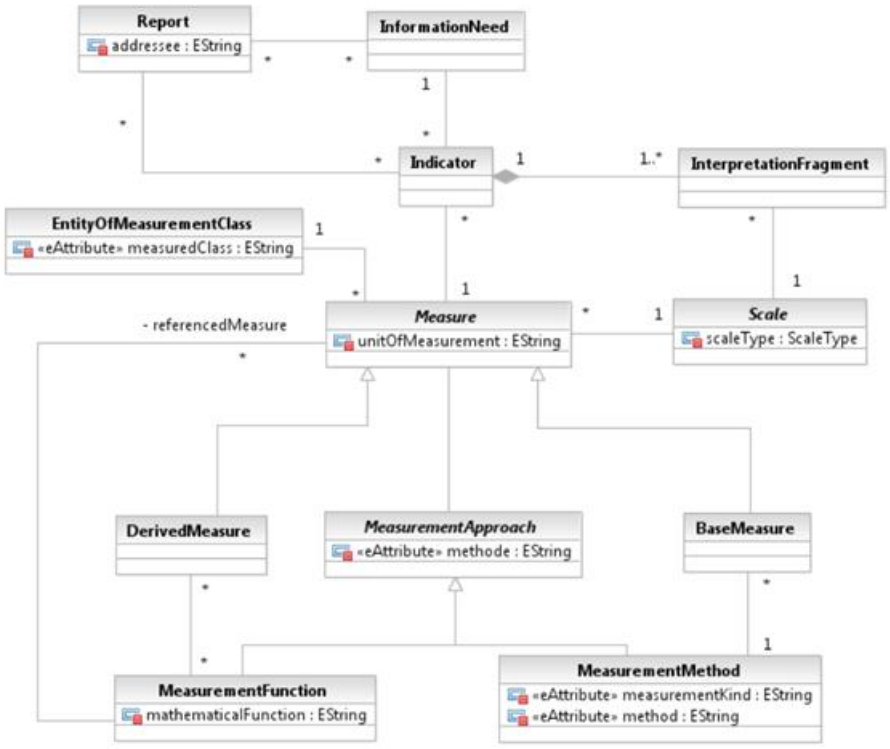

Figure 3.1: The core of metric meta-model

From the metric meta-model above, there are several main entities:

Information need Information need is the starting point of product or process measurement.

Interpretation fragment The interpretation aids provide an evaluation of specified attributes derived from an analysis model with respect to defined information needs. Interpretation aids are the basis for measurement analysis and decision making.

Indicators Indicators are formal interpretation aids that will be presented to stakeholders

Report The communication of the measurement results within the organizations is equally important as the collection of the metrics. For this purpose, a report is specified and with it the measuring results are displayed in a common document.

Entity of Measurement The first is the entity of measurement, which binds it to a portfolio of metrics belonging to the domain model. This can ensure that metrics are collected only from the entities for which they are defined.

Scale The second aspect that every metric requires is a scale. A scale is a set of values, continuous or discrete, or a set of categories to which an attribute is mapped. Every scale has a type, which specifies the values and also restricts the operations allowed for that type.

Measure The core aspect of the metric metamodel is measure entity. As mention in section (2.1.1.) measures are classify into 2 types, base and derive. Base measure involves no other entity, while derive measure obtained the value from other measures.

Measurement approach Each measure has a measurement approach to describe specific realization to obtain the measure's value. Different type of measure will have different approach; measurement method or measurement function. Base measure used measurement method expressed in a formal language or simple textual description, while derived metric used measurement function represented in a mathematical form with other measure as the mathematic variable.

A related work concerning the MeDIC system about variability is mentioned in Tavizon's works [Tav11] as a metric framework shown figure 3.2 below. The structure of metric frame is divided into two parts; upper part and underlying part. The upper part represents the core of metric as variation points and it consists of four categories: metric, interpretation aids, information needs and reporting. The underlying part forms the variant part, variability of the upper part.

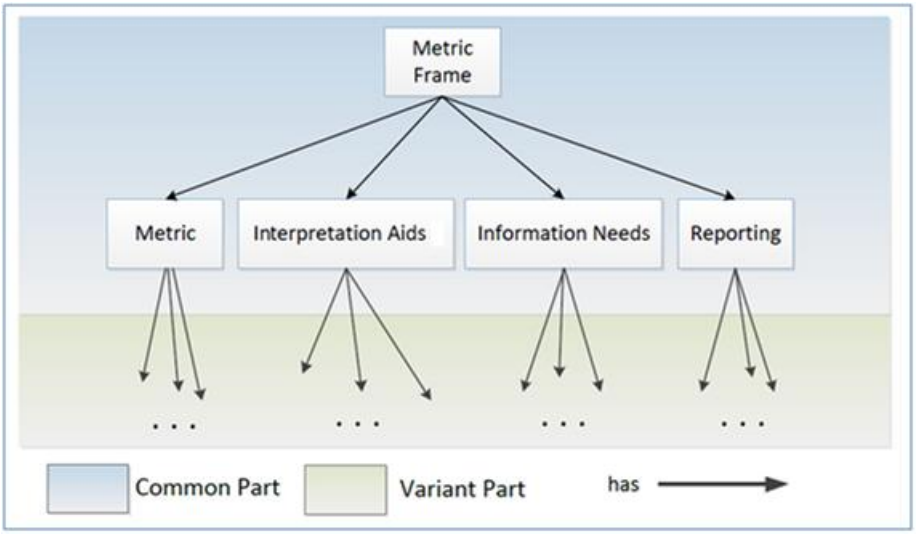

Figure 3.2: Metric frame structure

Variability analysis in Tavizon's work refers to variability in horizontal way, where the variants are determined as the possibility type of its variation point. For example, Tavizon analyzed variants of information need in the MeDIC system as all possibility of main categories of information need itself: quality, time, cost and price. Figure 3.3 illustrates the example of information need in graphical notation; shows variants quality, time, cost and price as alternative choice while the range [1,4] states that at least one variant must be selected and possibility to select all of the four variants.

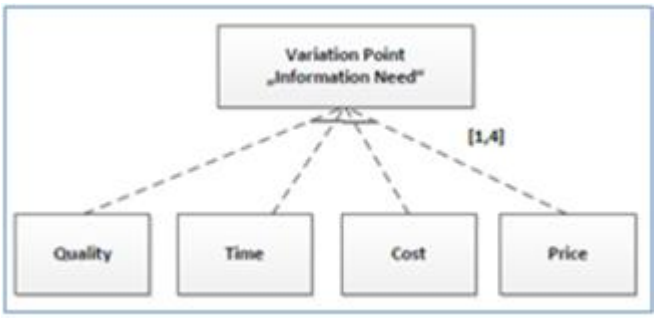

Figure 3.3: Variation point "'Information Need"'

Differ from the variability in horizontal way explained above; the variability in this master thesis will be focus on entity variability under the same system, which is MeDIC information system. Entity variability is the varying degree of entity specification. Some variation points as the result of entity variability analysis will refer to the same variation points shown in the figure 3.2, with different variants. Fetching the same example of information need variation point before, over the analysis phase (will be explain later on next section 3.3), the variants of information need are determined as the variation specification state of information need entity: described, formulated or categorized.

Variation Point Based on the variability source mentioned in analysis phase (section 3.2.1), several entities from metric meta-model are determined as variation points:

- Information Need
  Information need as the starting point of proposing metric is initiated with text description and over some states of specification information needs will evolve into implemented model.

- Metric/Measurement
  Metric measurement will evolve into different type of measure, either base or derived metric, while both types of measure are started with the same initiation form.

- Reporting
  Reporting activity is on the last phase of the metric system, to visualize and communicate the metric itself to the stakeholder. Therefore, differ-

ent type of visualization, interpretation or indicator of the report varies between projects.

In this master thesis, the variability modeling will only be focused on two main variation points: information need and metric measurement. Information need as the starting point, while measure is the core element of metric metamodel. Therefore, the modeling solution will be provided with applicable condition, to support the modeling of other entity variability in MeDIC system or in the other system.

Domain Scenario of Variation Points After defining two main variation points in the MeDIC system, the detail of problem domain scenario will be described in this subsection. Domain scenario of each variation point will explain all steps of the process behind the specification of the entity. From the scenario, variants for each variation point can be defined easier.

Information Need Domain Scenario

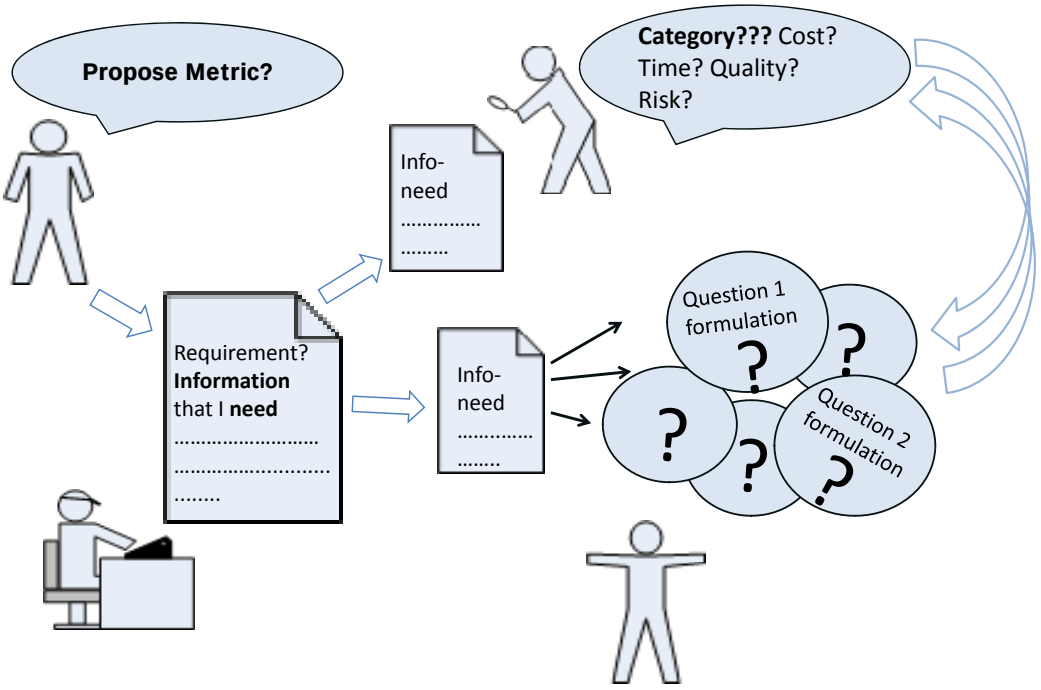

Figure 3.4: Information need domain scenario

Figure 3.4 as shown above illustrate the domain of information need entity specification. The process is started with initiation for proposing metric with textual description of all general information that is needed. The next step of specification is either categorization or question formulation, both of the steps have to be accomplished to get the completed entity of information need. Categorization is a step to group similar information needs with the existing category in the system, the main category on the current system are cost, time,

quality, content and risk; with several subcategories for each main category. Categorization aims to realize the reusable functionality, based on the same category to reuse existing metric. On the other hand, question formulation is a step to break down the rough description of information need into several detail questions. This step is carried out with formulating a question that will be answered by a metric that is proposed later on. The question for each information need description can be formulated more than one, in one time or in different step.

Metric/Measurement Domain Scenario

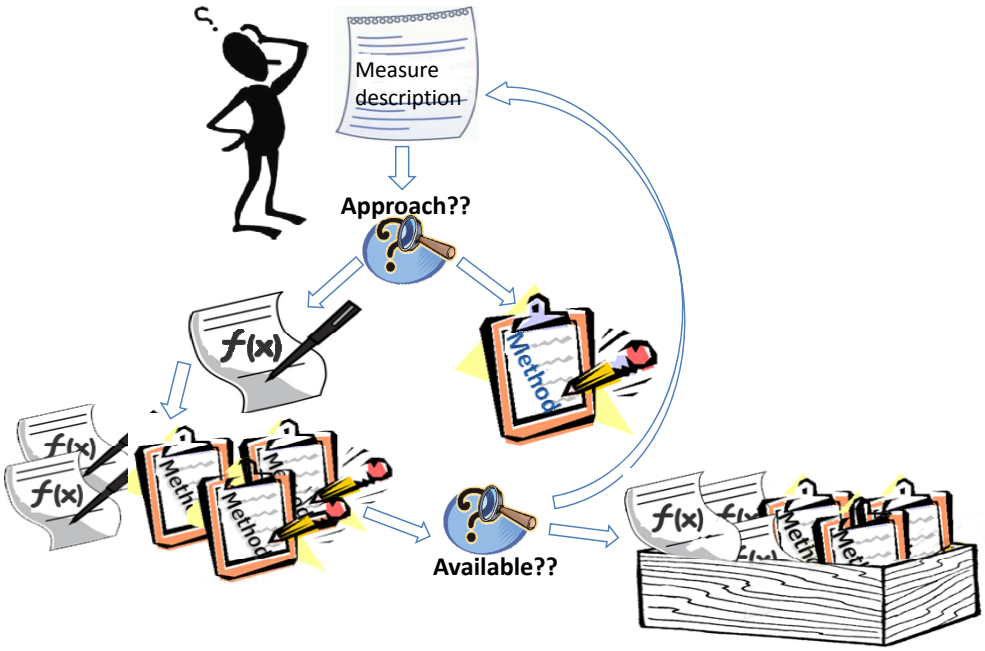

Figure 3.5: Metric measurement domain scenario

The second main variation point in MeDIC system is measure entity. As depicted in figure 3.5, the specification of the measure is started with a description as well, to describe broad information about the measure itself. The next step is to specify the measurement method, how to get the value of the entity. There are two different approaches of measurement, method approach and function approach. The type of approach will determine the type of measure; either it is a base measure or a derived measure. Both of the measurement approaches are first described in a rough description; a simple textual description for method measurement, or description in mathematical form for function measurement approach.

The mathematical function is formed from variables, and those variables are the value of another measure entity, therefore any measure that used function

approach are categorized as derived metric. Therefore, the next step in derive measure is to related all measures that mentioned in the function description. Measures used in the function can be either derived or base measure; for derived measure will recursively follow the same steps to break down the measurement approach specification. There are two possibilities to find the related measure entities, simply connect to existing measures in the system or create new measures to provide the value for the function if the measures are not available.

3.3 Modeling variability

3.3.1 Determine all variants for each variation point

Variant is variability point of the variation point. The commonalities of all related variants are reflected in variation point itself. In the metric information system, the main commonality of an entity is the initiation phase of entity specification; where an entity is initiated by textual description. From the scenario domains explained in the previous section, we will define variants for both variation points; information need and measure.

Information need

The process to identify complete entity of information need is divided into 3 steps: initiation, categorization, and formulation. Initiation is the first step to describe information need as textual description. Categorization is a process to group the information need based on the existing category in the system for reuse purpose. While formulation is a process to break down the information need from textual description into question form and for one information need description can be formulated into several questions. The state diagram of information need is depicted in the figure 3.6

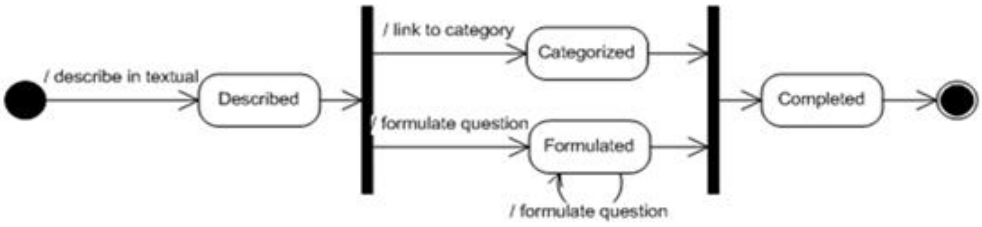

Figure 3.6: Information need state diagram

Based on the process to specify information need entity, the variants of information need are determined as the different state of the information need entity; initiated, categorized, formulated, or completed. Therefore, the first variability

mechanism proposed to model the entity variability of information need is state pattern; will be discussed later on section 3.3.2.

Measure

From the same initiation point, measure entity later on will evolve into different type of measure; base or derived. The type of measure depends on the measurement approach chosen to get the value of the measure itself. As mentioned before, there are two measurement approaches; measurement method and measurement function. From figure 3.5

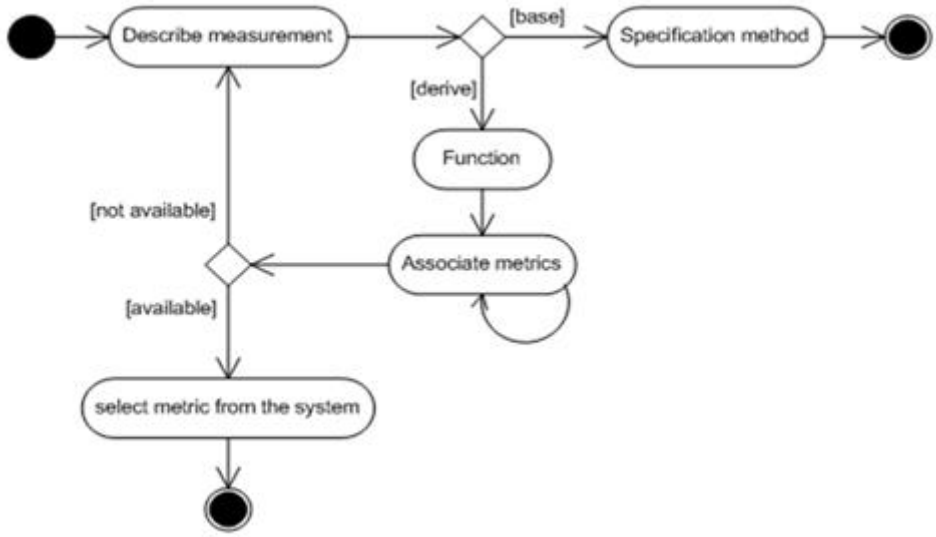

Figure 3.7: Metric measurement state diagram

3.3.2 A Set of Predefined Pattern

Patterns provide reusability and routine solutions to certain types of problems. Furthermore, patterns support the reuse of underlying implementation. Choosing a correct pattern for the appropriate problem will help designer to solve the problem faster with reuse successful designs and architectures. Several existing design patterns are referred to variability mechanism, to mapping the variability from requirements to implementation needs. A set of predefined patterns will be introduced in this section, to explain the structure of the pattern itself and the applicability for each pattern in variability mechanism. The set of predefined pattern is consists of strategy, state, adapter and decorator pattern. A solution pattern will be proposed to provide a way to implement entity variability in the MeDIC system. The proposed pattern can be incorporated into any system to build entity variability model, from which they can be derived.

Strategy Pattern

Strategy pattern is the design pattern most frequently used in the context of product family engineering. The idea of the strategy pattern is to make different algorithm variants, which are hidden in the common interface. The algorithm variants are derived from a default algorithm variant using inheritance. The concepts of encapsulation and inheritance can be used to implement design patterns that describe variability. Figure 3.8 depicted the structure of strategy pattern, as elaborated as below:

- Context; manages the data structures that a concrete strategy operates on.
- Strategy; defines the generic interface.
- ConcreteStrategy; provides the implementations of the different strategies. These operate on the data structures in the Context, and can be set dynamically

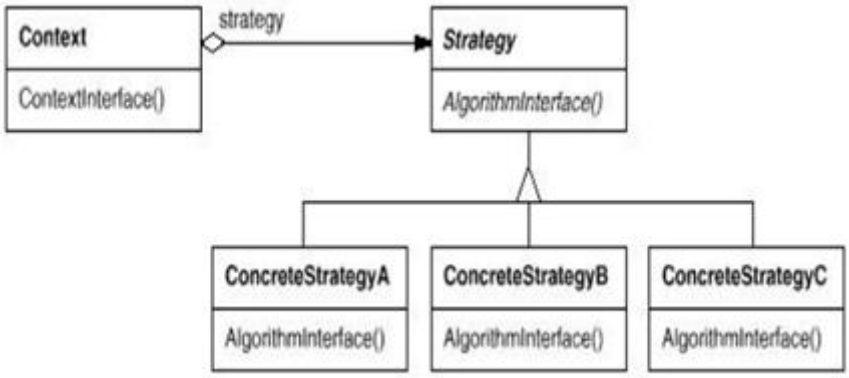

Figure 3.8: Stragegy Pattern

Strategy pattern define a family of algorithms, encapsulate each one and make them interchangeable. Applicability of strategy pattern is depend on two conditions; when many related classes differ only in their behavior or/and the need of different variants of an algorithm. The benefits of strategy pattern are:

- Provide an alternative to subclassing the Context class to get a variety of algorithms or behaviors
- Eliminate large conditional statements in implementation code
- Provide a choice of implementation for the same behavior

In the implementation, strategy pattern increase the number of the objects. Another condition as liability of strategy pattern is that all algorithms must use the same strategy interface.

State Pattern

State pattern has similar structure with strategy pattern, both pattern are example of composition with delegation. The main difference of those both patterns is one of intent; a strategy pattern encapsulates the algorithm while state pattern encapsulates a state-dependent behavior.

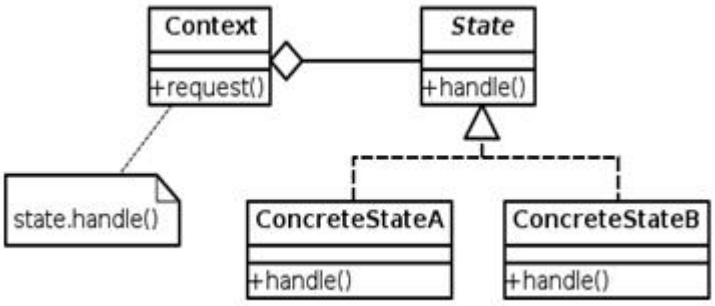

Figure 3.9: State Pattern

As illustrates in figure 3.9, state pattern has the same structure with strategy pattern; context class, state as interface class and concrete classes:

- Context; as the class that can have a number of internal states, defines the interface of interest to clients. The request() made on the Context will be delegated to state.handle().
- State; defines an interface for encapsulating the behavior associated with a particular state of the Context.
- ConcreteState; handle requests from the Context. Each concrete class implements a behavior associated with a state of the Context.

The same issue occurs on the implementation of state pattern; result in a greater number of classes in the design. But on the other hand, state pattern is applicable for design an object's operations that have large and multiple conditional statement that depend on the object's state. Moreover, an object's that depends on its state and must change its behavior at runtime depending on that state. To conclude the state pattern with variability concept, state pattern is used to design an object that has variant states and its possibility state transition, while the variants are decided on the runtime by user.

Decorator Pattern

The decorator pattern is also known as wrapper, since it describes a pattern in which an object can be enclosed by another object, with the enclosing object controlling input and output. By using this pattern, programmers can dynam-

ically attach functionalities to an object without modifying its internals. In a product family system, variants can be described as objects that have different features. Decorator pattern allow an object to add many features and functionalities dynamically, especially when the combination of features could not be predict in design time.

Figure 3.10 depicts the structure of decorator pattern, contains of four main components as describe below:

- Component; defines the interface for objects that can have responsibilities added to them dynamically.
- ConcreteComponent; defines an object to which additional responsibilities can be attached.
- Decorator; maintains a reference to a Component object and defines an interface that conforms to Component's interface.
- ConcreteDecorator; adds responsibilities to the component.

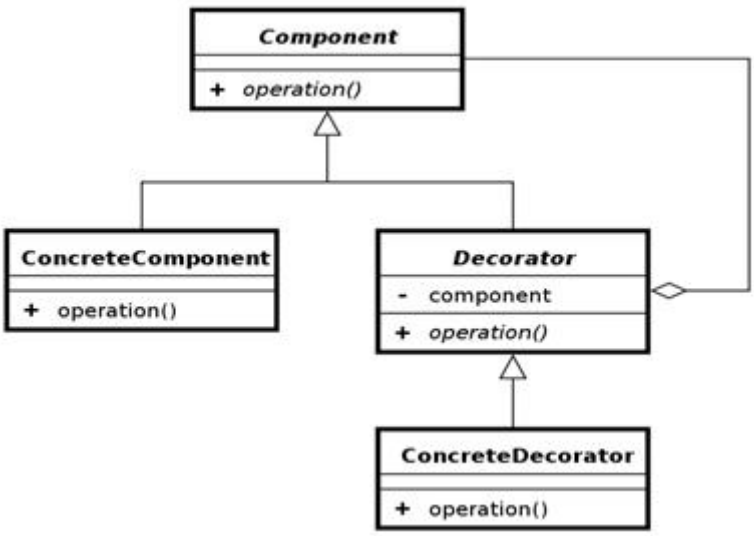

Figure 3.10: Decorator Pattern

Decorator pattern provide a flexible alternative to subclassing with combination of composition and inheritance. Furthermore, the advantages of decorator pattern will be listed below:

- Add flexibility and extensibility, user gets to decide how many decorators are applied dynamically and in what order
- Can nest decorators recursively, allowing unlimited added responsibilities
- Elimination of unnecessary inheritance hierarchies, fewer classes than with

static inheritance

As an issue, decorator can result in many small objects in the design and the overuse object will lead to complexity. Application of decorator pattern is concern on adding responsibility to individual object transparently; encapsulating responsibility again without affecting other object; and extending classes without explosion of subclasses.

Adapter Pattern (simple decorator)

Adapter pattern also known as wrapper; while decorator wraps a class with another class to add functionalities, adapter pattern wraps a class with another class to convert interface of existing class into an interface that is compatible with an otherwise incompatible client class. Adapter changes the interface of a class that is being presented, with no new functionality is created in adapter class. The main purpose of adapter pattern is to make the classes that have different interface to work together, preserving the reusability of the existing class.

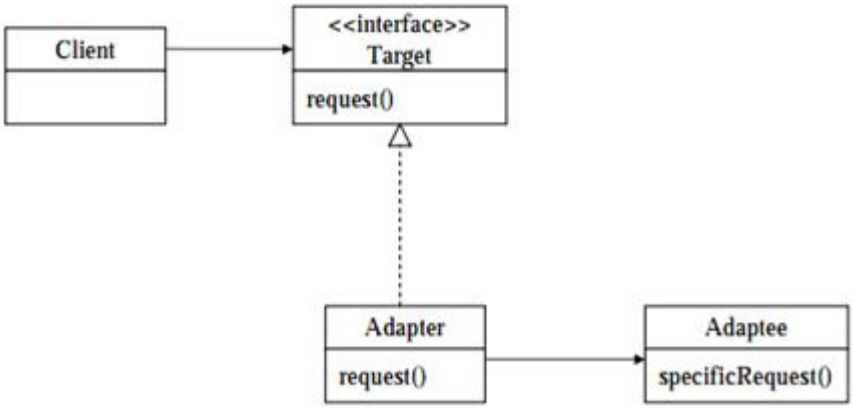

Figure 3.11: Adapter Pattern

The structure of adapter pattern (figure 3.11):

- Client; client see only the target interface
- Target; as the expected interface
- Adapter; implement the Target interface, adapter is composed with the Adaptee
- Adaptee; all requests get delegated to the Adaptee

There are two forms of adapter pattern; object and class adapters. Object adapter is the basic pattern describe in figure 3.11 with using composition to

adapting the adaptee, while class adapter uses multiple inheritance to subclass the Target and the Adaptee. The best application of adapter pattern is for collaboration between classes with incompatible interface. Implementing an adapter may require little work or a great deal of work depending on the size and complexity of the target interface.

3.3.3 Preliminary Works

This section will introduce several preliminary works to find the best solution for modeling entity variability. Start with the understanding of variability notation in feature modeling, adaptive object model as an alternative that represent all information as metadata, until finding design pattern as the best mechanism to model entity variability.

Feature Modeling

Feature model is the most common representation for variability in product software line and visually represented by means of feature diagrams. Feature models were first introduced in the Feature-Oriented Design Analysis (FODA) and since then, a number of extensions have been proposed. As variability related, this work starting with understanding about feature model and its notation that is used generally for variability presentation. Nevertheless, variability in this master thesis is focus on entity variability, variability that brings all variation of specifications of one entity not variability of all final products in a product family. Furthermore, the feature model provides only notation for modeling variability, thus feature modeling is not a solution that can be used and modified to propose general structure for entity variability in the MeDIC system. The purpose of this preliminary work is rather to get the first impressive about variability in general usage. Later on, the implementation of variability in this work might have different interpretation with general variability in software product line engineering application.

Adaptive Object Modeling (AOM)

Flexible and adaptive modeling to handle the specification change of an entity is the direction of entity variability modeling that is required in this chapter. Adaptive Object Modeling (AOM) is one of modeling for facing requirement change within application domain, where the business rules are changing rapidly. An Adaptive Object Model is an object model that provides 'meta' information about the domain so that it can be change at runtime . AOM create an object design (meta-model) that describes the domain objects which includes attributes, relationships, and business rules as instances rather than classes. The architectural pattern of AOM was made of many smaller patterns to build very flexible

systems; such as type object, properties, strategy, type-square, and interpreter pattern. Figure 3.12 shows the common design structure of Adaptive Object Model with combination of several patterns within.

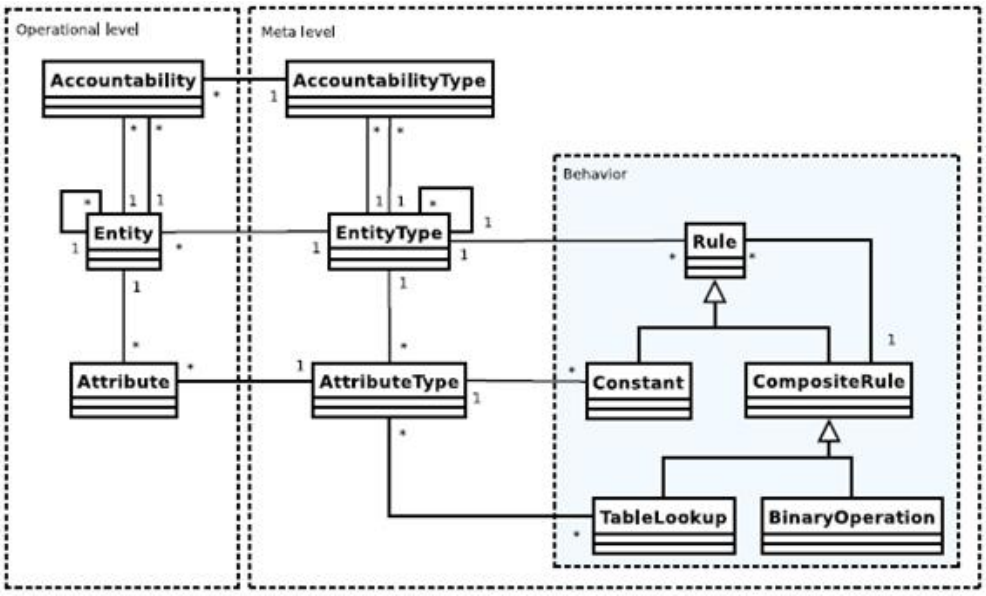

Figure 3.12: Basic design of Adaptive Object Model

By creating a new description and a new instantiation to satisfy each new requirement, system became more adaptive with domain changes where the changes do not require recompiling the system. Furthermore, AOM allows user to change the business rules in order to extend the system. With the flexibility provided by AOM, this model might meet requirements of our entity variability model, to evolve the specification of an entity dynamically. However, AOM has several disadvantages at implementation point that must be considered before apply this model to metric meta-model concern with variability issues. Developing AOM requires skilled human resources, because it is hard to understand and maintain. The startup cost to develop AOM is expensive; moreover the model can have poor performance. In order to apply this object model in the MeDIC system, high effort will be required to rebuild the entire system. Rebuild the entire system in order to model specific variability points in metric meta-model will require too much effort and cost, therefore this object model is not an appropriate solution for our case.

State Pattern

Over several existing variability mechanisms, patterns were chosen as a solution to build general structure for entity variability. Patterns provide reusability to handle the similar modeling problem, which is an issue to model variability of an entity in this work. From a set of predefined patterns for variability that are

described in section 3.3.2, state patterns are the first pattern that we tried to apply for modeling entity variability of information needs.

Specification process of information needs as a complete entity is following three steps; from textual description, categorization, until question formulation. Therefore, the information needs present three states in its life cycle; 'describe', 'categorized', and 'formulated'. With variability focused on the state of entity, using state pattern as the pattern to present the state of an entity seems to be applicable to modeling entity variability of information needs.

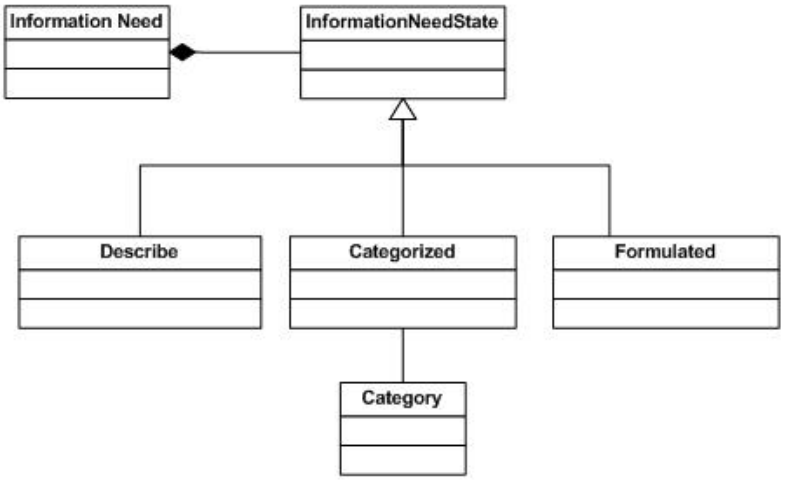

Figure 3.13: State pattern of Information Need

Figure 3.13 shows a result model for the entity variability of 'InformationNeed' through a mapping of entity variability problem into state pattern structure as explained in figure 3.9. 'InformationNeed' represents the Context class as the state base class that will maintain an instance of three states of information need entity that defines the current state. 'InformationNeedState ' associates 'InformationNeed' and all the concrete states as common interface for encapsulating the behavior of all the concrete states and controlling the state transition. With implementing the same interface, the states are interchangeable . From the variation point analysis (3.3.1) of information need entity, process of entity initiation until the complete entity is formed we have identified three states of information need; 'Describe', 'Categorized', and 'Formulated'. All of the states are modeled as ConcreteState classes as shown in the figure 3.13. All the states will handle requests from the 'InformationNeed'. Each ConcreteState provides its own implementation for a request. In this way, when the 'InformationNeed' changes the state, its behavior will change as well. For example, when the current state 'InformationNeed' entity change from 'Describe' to 'Categorized', the behavior and property will change as well that will allow the information need entity to be categorized by associate it with 'Category' class.

With state pattern applied in the model, information need entity is become more flexible to be implemented, easier to add new states in the case of the

process to create complete entity, and avoid the inconsistent states. However, some withdraws of state pattern have to be observed. Class explosion might occur from the ConcreteState classes. Although in information need entity all the states have been determined into three states, process to identify complete entity might develop and need the increment of other states. In the other case, another entity might have numerous states to identify a complete entity, and every state will one class, which is ConcreteState class to be model in the structure. Variability modeled by state pattern is focused only on the state of entity, while the metric entities exist in the MeDIC system are various. The other core entity in MeDIC system that is closely related with variability is 'measure' entity. Variability in 'measure' entity occurs as varying types of the evolved entity. 'Measure ' entity is identified from an textual description and later on will develop into 'base' or 'derived' metric based on measurement approach chosen for every metric. State pattern as variability mechanism barely apply to model variability in measure entity. Therefore, state pattern could not be applied to proposed general variability modeling in the MeDIC system. In the next section, another variability pattern from a set of predefined pattern (3.3.2) is proposed as a solution to model both core entities in the MeDIC system.

From several preliminary works presented, we hope that next researches that are dealing with variability, especially in modeling entity variability might get a picture on how and why the solution variability is proposed in this work. Variability is an extensive topic applied in current industrial. Many researches have been published to work through variability in various perspective and purpose. We hope that this work will render another research of modeling variability in software engineering.

3.3.4 Applied variability solution pattern for each variation point

Variability solution pattern proposed to model the variability in the MeDIC system is adopted from decorator pattern, one of predefined pattern that are explained in section 3.3.2. A concept or model that we are looking for to model variability in the metric system is a model that can flexibly present the metric as the reality. The user can propose and refined the metric dynamically. This concept is still difficult to be modeled. However, the proposed model from decorator pattern can reflect the current need in our system. Decorator pattern allow an object to add many features and function dynamically, especially when the combination of features could not be predicted in design time. As we mapping to our core metric entity, decorator pattern can represent the variability.

In this section, decorator pattern will be applied to model variability for both core entities of metric on the MeDIC system: Information Need and Measure entity. Compare with state pattern explained in the previous section, decorator pattern provides more flexibility with various decorators and dynamic order/sequence to attach the decorator. The term of decorator in this section will refer to additional specification of an entity as result of entity refinement process to

attain the complete form of the entity itself, such as functionalities, properties, attributes, etc. For example, the decorators of Information Need entity are the category and question formulation as the additional properties, include all functions attached. Those decorators of Information Need entity are attached later on to specify the entity to be more detail and measureable.

Information Need Variation Point

The main structure of decorator pattern is divided into two parts, the concrete component and the decorators. Concrete component is the main component of the entity, that will be exist in all possible variant of the entity, whereas the decorators are the components that causing the variation between entities. The variation can be the different type of decorators that are attached to the entity or the amount of decorators for each entity. For Information Need entity, the concrete component is the initiate form of the entity which is the abstract textual description. All variations of information need entity are started from the same initiation form, and later on will evolve to different form. On the MeDIC system, almost all of the entities, especially those that have variability, are started with textual description, therefore, it can be assumed that concrete component in decorator structure of all entities on the MeDIC system is the initiate form of the entity itself.

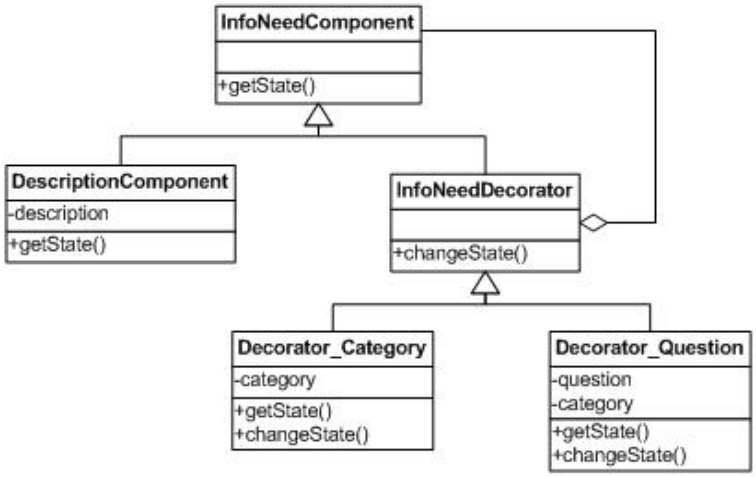

Figure 3.14: Decorator pattern of Information Need

Figure 3.14 presents the mapping result of decorator pattern to Information Need entity. Component class as the interface is represented by the entity itself, 'InfoNeedComponent' class and concrete component class is represented by 'DescriptionComponent' class. All of the information need entity object created will include 'DescriptionComponent' as the first component for the entity, to describe initiation of information need in textual description. All classes that are derived from 'InfoNeedDecorator' abstract class are the decorators, those decorators will wrap the initiate entity object with new component and refined the entity. There are two decorator classes for information need entity; Dec-

orator_Category and Decorator_Question. Decorator_Category is a class to reflect categorization of Information Need, while Decorator_Question is a class to describe the formulation of Information Need's question.

The structure of decorator pattern provides flexibility to add more property that can reflect the variability of the entity. Entity variability in this thesis is focus on varying degree of specification that is represented by different properties and functions in different state of entity specification. The decorator attached to entity will change the specification dynamically.

Measurement Variation Point

Differ from Information Need entity, Measure entity is barely modelled with state pattern. State pattern is focused on the state of an entity, but the variability of Measure entity is a variation point of different type of the evolved entity. The state could not be reflected well, moreover, all the evolved types of Measure entity are started with the same initiate form. When the user proposes the measure, the first entity is specified as a textual form to describe the measurement of the entity. The type of the entity in an initiation stage could not be known until the next specification step is committed; to specify the entity with a measurement approach. Later on, the measurement approach can be specified again with related Measure entity that is used in the measurement approach.

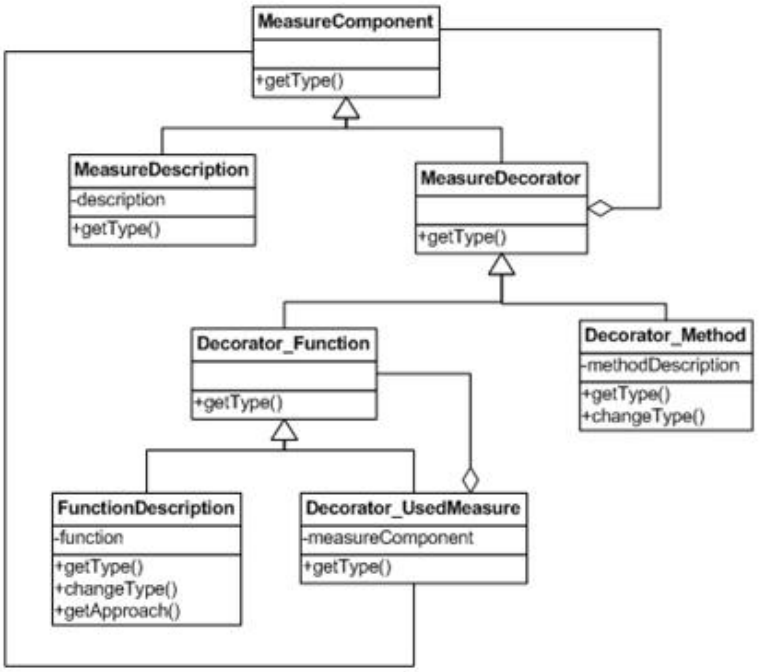

Figure 3.15: Decorator pattern of Measure

The figure 3.15 illustrates the structure of Measure entity with nested decorator

pattern applied on it. MeasureComponent class is the interface of the component. MeasureDescription class present the initiate form of the entity as the concrete component class in decorator pattern structure. Decorators of MeasureComponent are Decorator_Function and Decorator_Method class. Both of the decorators are measurement approach that is used to specify the Measure entity and determine the type of the entity. If the method approach is attached to the entity the type of Measure entity will be called as base metric, while entity with function measurement approach will be addressed with derived metric (because the value is derived from other entity's value as an calculation result). Method approach is a plain measurement description to explain how to get the value of the entity. While function approach is a functional statement that involved another Measure entity value as the source of the calculation. Therefore, the function decorator in Decorator_Function class can be decorated more with FunctionDescription and Decorator_UsedMeasure to specify the function itself and Measure used in the function. For this purpose, nested decorator is applied to the Measure entity model.

3.4 Variability mechanism (solution pattern)

In the previous section pattern has been discussed, especially for state and decorator pattern for modelling variability of core metric entities. Pattern in this thesis refers to variability mechanism, chosen from several variability mechanisms (2.2.3.), that is proposed to enhance MeDIC meta-model. From two core entities in the MeDIC system, decorator pattern are prove to be able to model the variability, and in this section the general solution pattern based on decorator pattern to model entity variability will be presented.

Figure 3.16 shows the general solution pattern for entity variability. Mapping from decorator pattern structure, Entity defines the interface for each entity. DescriptionComponent is the concrete component for each entity in abstract textual description form; the object that will be initiated first and extends the Entity. New property/component will be added dynamically to the DescriptionComponent to specify the entity. All decorators present component to specify the entity, with new state, new component that will wrap the entity, or new behaviour. The decorators are derived from EntityDecorator class that implements the same interface with Entity. The decorators can add new method or extend the state of the entity, and each decorator has an instance variable for the entity it decorates. Those decorators that are used to wrap the entity will change the specification degree of the entity and cause the variability between entities. The structure can be nested to model a decorator component that can be decorated more with another decorator, as showed in the Measure entity model in section 3.3.4.

This solution pattern is mainly aimed for modelling variability of an entity in the

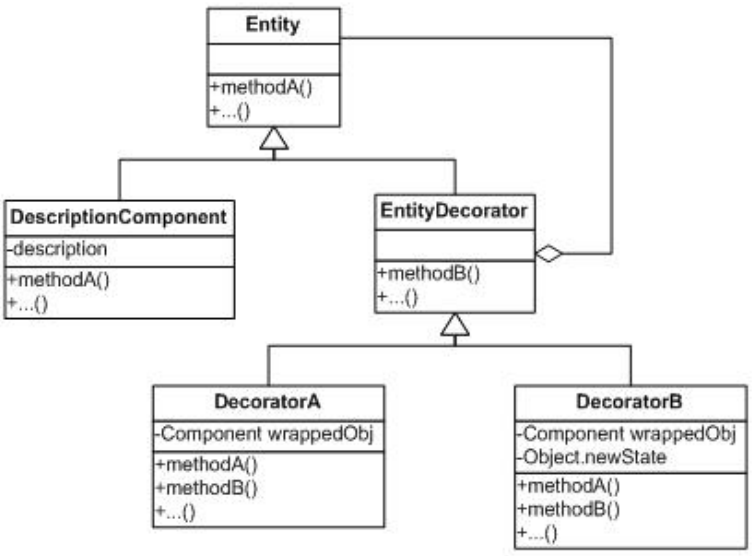

Figure 3.16: Solution pattern for entity variability

MeDIC system. Metric's entities in the MeDIC system are specified over several process steps and each step will result a variant of the entity. Moreover, the same process step might create different variant from different decorator applied to entity or different amount of the decorators. Therefore, specific model need to be present to cover this specific purpose.