
# Variability modelling in divide and conquer method

**By**
**Ahmad Fouaad AL-Jassim AL-Sultan**

**Supervisor**
**Prof. Said Ghoul**

**This Thesis was Submitted in Partial Fulfilment of the Requirements for the Master's Degree in Computer Science**

**Deanship of Academic Research and Graduate Studies**
**Philadelphia University**

**2014**

جامعة فيلادلفيا

نموذج تفويض

أنا احمد فؤاد الجاسم السلطان ، أفوض جامعة فيلادلفيا بتزويد نسخ من رسالتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبها.

التوقيع :

التاريخ :

**Philadelphia University**

**Authorization Form**

I am, Ahmad Fouaad Aljassim Alsultan, authorize Philadelphia University to supply copies of my thesis to libraries or establishments or individuals upon request.

Signature:

Date:

# Variability modelling in divide and conquer method

**By**
**Ahmad Fouaad AL-Jassim AL-Sultan**

**Supervisor**
**Prof. Said Ghoul**

**This Thesis was Submitted in Partial Fulfilment of the Requirements for the Master's Degree in Computer Science**

**Deanship of Academic Research and Graduate Studies**
**Philadelphia University**

**2014**

Successfully defended and approved on _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

| Examination Committee Signature | | Signature |
|---|---|---|
| Dr.<br>Academic Rank: | , Chairman. | _ _ _ _ _ _ _ _ _ _ _ _ |
| Dr.<br>Academic Rank: | , Member. | _ _ _ _ _ _ _ _ _ _ _ _ |
| Dr.<br>Academic Rank: | , Member. | _ _ _ _ _ _ _ _ _ _ _ _ |
| Dr.<br>Academic Rank: | , External Member. | _ _ _ _ _ _ _ _ _ _ _ _ |

# Dedication

First of all I thank Allah the almighty for giving me the strength and knowledge to finish this work, I dedicate this work to *my family my father, My mother, My brothers, My sisters and my uncles, also to my friends*, they've been there for me whenever I needed.

*Ahmad fouaad alsultan*

2014

# Acknowledgment

(بسم الله الرحمن الرحيم وَمَا أُوتِيتُمْ مِنْ الْعِلْمِ إِلاَّ قَلِيلاً)

It would not have been possible to write this master thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Above all, I would like to express my thanks and sincere gratitude for who has guided me through my study and my thesis work; my supervisor *prof. Said Ghoul*, for giving the wisdom, strength, support and knowledge in exploring things.

I would like to thank my family members; *My father, My mother, brothers, sisters, uncles and my friends* for giving me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

Also, I am grateful for those who supported me and encouraged me in any way; *my teachers* at Philadelphia University.

*Ahmad Fouaad Alsultan*

*2014*

# Table of Contents

# List of Tables



# List of Abbreviations

| Abbreviation | Full Name |
|---|---|
| MM | Meta Model |
| UML | Unified Model Language |
| FD | Feature Diagram |
| D&C | Divide and Conquer |
| BNF | Backus–Naur Form |

# List of Figures

# Abstract

Divide and Conquer (D&C) is a very broad problem solving pattern, used in several areas: mathematics, computer science, physics, engineering, etc. Consequently it constitutes an active topic for varied researches. Some researches deal with the approach enhancements (parallelization, distribution, adaptation, etc.). Others deal with its application to solve various problems in different application domains and its evaluation. But despite this active and intense research, until now: (1) no general D&C method variability meta model, that covers several classes of problems in different domains, (2) no formal methodology supporting this variability, (3) no formal problem specific variation selection and instantiation process, and (4) no modelling languages suitable features have been proposed.

However, rare patterns for specific problems were developed. Developing a broad divide and conquer pattern requires the application of abstraction techniques where the meta modelling is the best candidate. Its intensive and continuous enhancement researches, power, broad use, and formalization capacities with its supporting languages like the Unified Modelling Language (UML) and Feature Diagram (FD), make it an effective model at the top rank.

This thesis, aims to overcome the above four D&C researches insufficiencies by proposing a rich and general D&C method development methodology based on variability modelling in D&C method and on a formal variation selection and instantiation process. Obtained result states clearly the suitability of FD formalism instead of UML for this kind of meta modelling and identifies UML possible enhancement that may generalize it to support methods variability meta modelling.

# CHAPTER ONE: INTRODUCTION

## 1.1 Preface

Problem solving (Koripadu, 2014; Rebori, 1995) is one of the important process in computer science domain. Problem solving has many steps defining solving process model: the first step is defining the problem. The second step is identifying and defining a root case. The third step is generating alternative solutions. The four step is evaluating the alternatives. This Problem solving model has been used for variable problems.

Divide and conquer is one of the applications of problem solving methods. (Figure 1-1) it is a very broad problem solving pattern practically used in all complex scientific areaes: mathematics, computer science (in all area), physics, engineering, etc. It processes by recursively braking the problem into sub problems until reaching small cases for which small solutions exist, then combining these solutions in a way to carry out the complete solution (Chow, 2013; vander, 2012). Consequently, it constitutes an active challenging topic for varied researches (Lopez-Ortiz, 2014; Jin, 2012; vander, 2012).

Many problems in computer science can be solved iteratively or recursively. Divide and conquer is a recursive method defying a general design pattern from which specific problem solving might be generated.

Scarce research works have been conducted in divide and conquer meta modelling. An earlier attempt has used a design pattern technique for the most general divide and conquer model (Francés, 1998). Later, the Unified Modelling Language (UML) has been used for modelling sorting algorithms (Rahmani, 2010) but no one general Divide and Conquer Meta model, that covers several classes of problems in different domains, has been yet proposed.

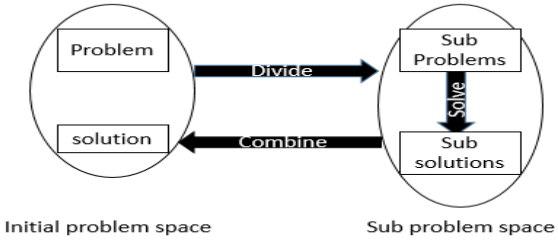



Figure 1-1. Divide and conquer method.

Divide and conquer design pattern may vary from one problem to another and modelling this variability is a challenge in the domain. Developing a broad divide and conquer pattern requires the application of abstraction techniques where the meta modeling (Gitzel, 2005; K¨uhne, 2006; Williams, 2013; Zuniga, 2013) is the best candidate. Effectively, a metamodel is a general abstraction from that other more specialized abstractions (Ehrig, 2009) and specific instances (Hao Wu, 2012) may be generated. Its intensive and continuous enhancement researches (Sprinkle, 2010), capabilities (Ma, 2013; Witherell, 2013), and formalization capacities (Henderson-Sellers, 2012; Giacomo, 2011), and its supporting languages like UML (Byrne, 2013) and feature diagrams (Kang, 1990), make it at the top rank.

## 1.2 Research Context

The idea of this thesis is to develop a meta model for divide and conquer. So its research context is about meta modelling, and divide and conquer methods.

Some researches deal with the divide and conquer approach enhancement (Lopez-Ortiz, 2014; Mateos, 2013; Hijma, 2008). These enhancements were generally proposed for supporting features like parallelization, distribution, synchronization, and more specific problem constraints. These related enhancements were separately developed but were not modelled in a structured abstract way, illustrating relations between them and allowing more effectiveness, comprehension, and reuse for building furthers ones. However some specific design pattern were presented (Francés, 1998; Rahmani, 2010).

A design pattern was presented by Javier and Julio (Francés, 1998), this design pattern does not deal with variability in problems, it is only a fixed method defined with UML notation.

An object oriented framework was presented by Rahmani and his colleagues (Rahmani, 2010) for modelling variability of sorting algorithms. All sorting algorithms are instances of the class “Asorter” only by defining the functions Split and Join. This framework is limited to sorting problems. So, it doesn’t deal with variability of problems, but it is a specialized and limited case of a broad divide and conquer method.

## 1.3 Problem Statement

From previous works, there are many challenges that might be defined:

- Methodology: there are many general patterns for divide and conquer method but they do not have methodology guiding their use.
- Meta modelling divide and conquer: the general pattern does not address variable cases such as empty combine, parallels or sequential schemas, empty small solutions, memorization, etc.
- Formalization: The used modelling notation in previous works is a Unified Language Model (UML) notation without adaptation evaluation. There are other languages in meta modelling that may be used in this case like Feature Diagram (FD).
- Instantiations process: there is not process to define instance of general pattern in actual divide and conquer method general pattern.

## 1.4 Motivation

The proposed research in this thesis was motivated by the following:

- Absence of a methodology that guides divide and conquer problem solving method.
- Absence of a broad meta modelling that deals with large cases of divide and conquer method.
- The limitation of the used notation to UML.
- Absence of instantiations process to generate instance from the divide and conquer general pattern.

## 1.5 Contributions

This work aims at designing a divide and conquer methodology based on a broad meta modelling:

- Providing a methodology to support general pattern divide and conquer method meta modelling.

- Designing a broad divide and conquer method based on meta modelling technique leading to a high abstraction meta model.
- Using UML and Feature Diagram as research and evaluate the adaptability of each one.
- Proposing a technique process for generating instance algorithm, specific to problem solving from the proposed divide and conquer meta model.

## 1.6 Thesis layout

The thesis starts with introducing the research problem in chapter one, related work in chapter two, then representing divide and conquer variable cases in chapter three. The proposed contributions solving some identified challenges in chapter four, and the evaluation of the conducted research in chapter five. .

# CHAPTER TWO: RELATED WORK

## 2.1 Introduction

This chapter presents some significant previous works in divide and conquer and introduces concepts of Meta Modelling (MM), Unified Model Language (UML) and Feature Diagram (FD).

Divide and conquer was approached by some researchers in the last years. Some had researches dealt with the divide and conquer approach enhancement (Lopez-Ortiz, 2014; Mateos, 2013; Hijma, 2008). A Design pattern was presented by Javier and Julio (Francés, 1998), this design pattern does not deal with variability in problems. An object oriented framework was presented by Rahmani and his colleagues (Rahmani, 2010) for modelling variability of sorting algorithms.

The Meta Modelling (MM) concepts are important domain of research: (1) in general abstraction (Sprinkle, 2010; Hao Wu, 2012; Gitze, 2005), (2) in specialized abstractions, (3) in supporting languages like UML (Byrne, 2013), and in Feature Diagrams (FD) (Kang, 1990). This makes it at the top rank.

## 2.2 Divide and conquer current modelling approaches

Divide and conquer is a method of problem solving, it is a very broad problem solving pattern practically used in all complex scientific areas.

Two researches presented divide and conquer (Francés, 1998; Rahmani, 2010) with specific model.

Javier and Julio (Francés, 1998) presented a design pattern for a high abstract divide and conquer, the structure of this design patterns, modelled with UML notations, is presented in (Figure 2-1) (Francés, 1998).

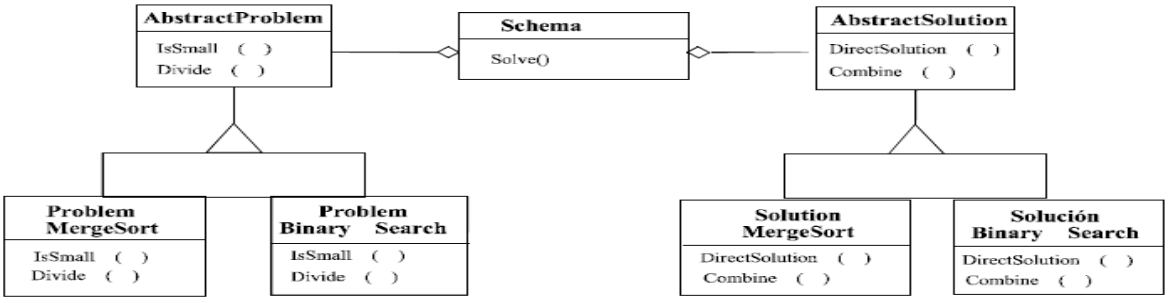



Figure 2-1. Structure of divide and conquer.

The schema (of divide and conquer) that composed of two components: (1) the Abstract Problem, containing the definition of the small problems (Is Small), for which a direct solution exist, and the composed problem (Divide), its solution which will be obtained from combining solutions of smallest problems. (2) Abstract Solution contains the definition of small solution associated with small cases sub-problems, and the definition of combine composing a global solution for small cases solutions. The work also presents the use of UML interaction diagram, and the sequence control in the schema. Specific algorithms solving particular problems, are instantiated from this design pattern, by specifying the generic parts: IsSmall, Divide, DirectSolution, Combine, sequence diagram.

An example, the (Figure 2-2), presenting the sum of array elements.

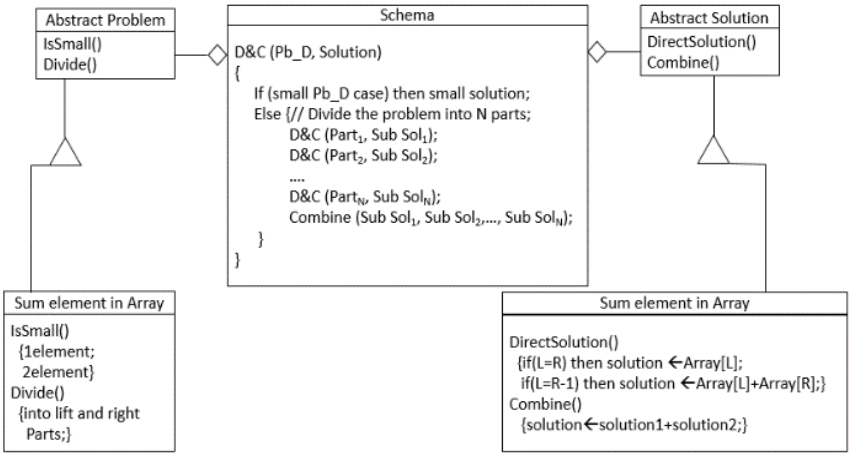



Figure 2-2. Sum array in divide and conquer.

However, this general design pattern does not specify cases where combine is empty, the schema is parallel or sequential, top down or bottom up, domain specific, etc. So, the knowledge reuse is poorly limited to the general structure of the design pattern.

Rahmani and his colleagues (Rahmani, 2010) have presented an object oriented framework for unifying different sorting algorithms as in (Figure 2-3) (Rahmani, 2010).

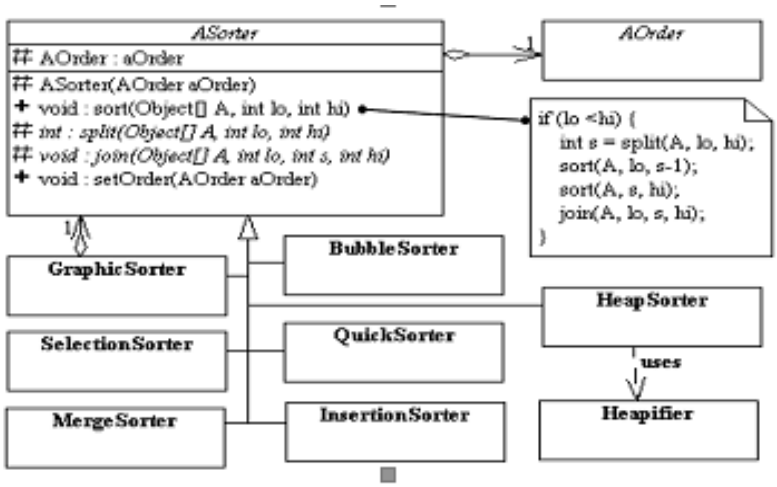



Figure 2-3. Framework unifying different sorting algorithms.

All the sorting algorithms are instances of the class "Asorter" only by defining the functions Split and Join. This framework is limited to sorting problems. This class is used in sort case only, so it is not useful for others cases.

An example, the (Figure 2-4), presenting the quick sort of array elements.

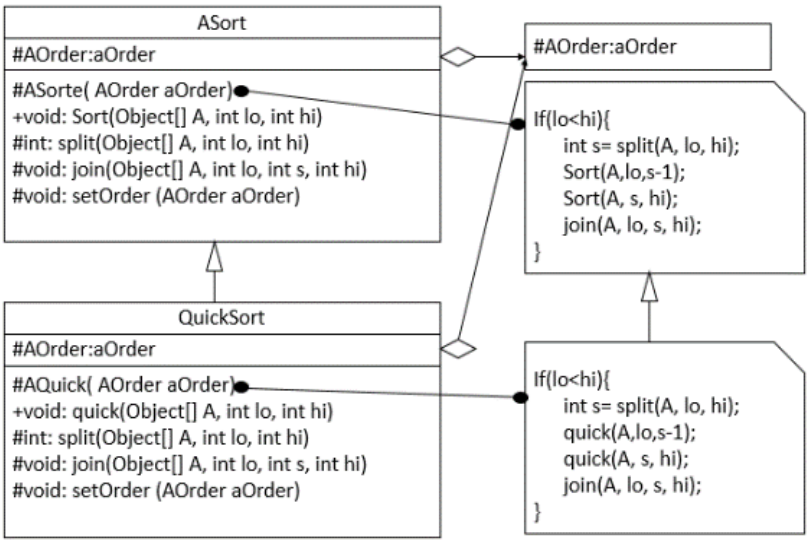



Figure 2-4. Quick sort algorithm.

## 2.3 Meta model concept

Meta Modelling (MM) (Sprinkle, 2010) is a rule to represent the system or model in general abstraction through using meta modelling language, and how to generate instance from general model.

A model is represented by meta model techniques (Sprinkle, 2010; Hao Wu, 2012; Gitze, 2005), it is a powerful technique. A Meta Modelling was represented by four layers: the first layer consists of meta-meta model that describe and define meta model layer. The second layer consists of meta model that define language to describe model layer. The third layer consists of model that. The fourth layer consists of instance from a model in the third layer as in (Figure 2-5) (Sprinkle, 2010).

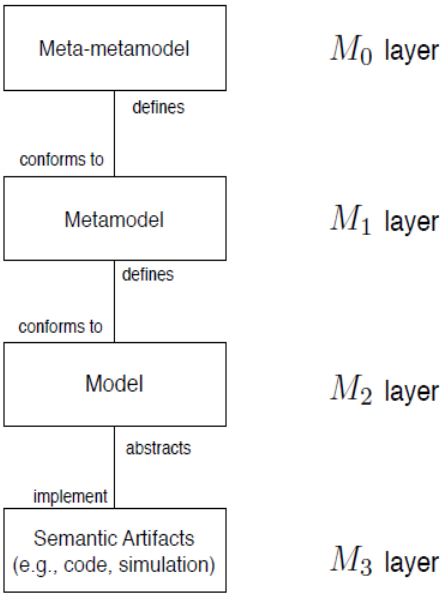



Figure 2-5. Meta model layers.

Now will be explain the layers in meta model architecture (Clark, 2008):

- Meta-meta model layer: this layer describes the characteristic in meta model layer like modelling languages.
- Meta model layer: this layer contains in language like UML, FD, classes, attributes, and operations.

- Model layer: this layer contains application object-oriented system, and the table definitions of a relational database.
- Instance layer: this is instance of object oriented class, instance of method or part of table database.

An example, the (Figure 2-6), presenting the divide and conquer Meta Modelling layers for sorting example.

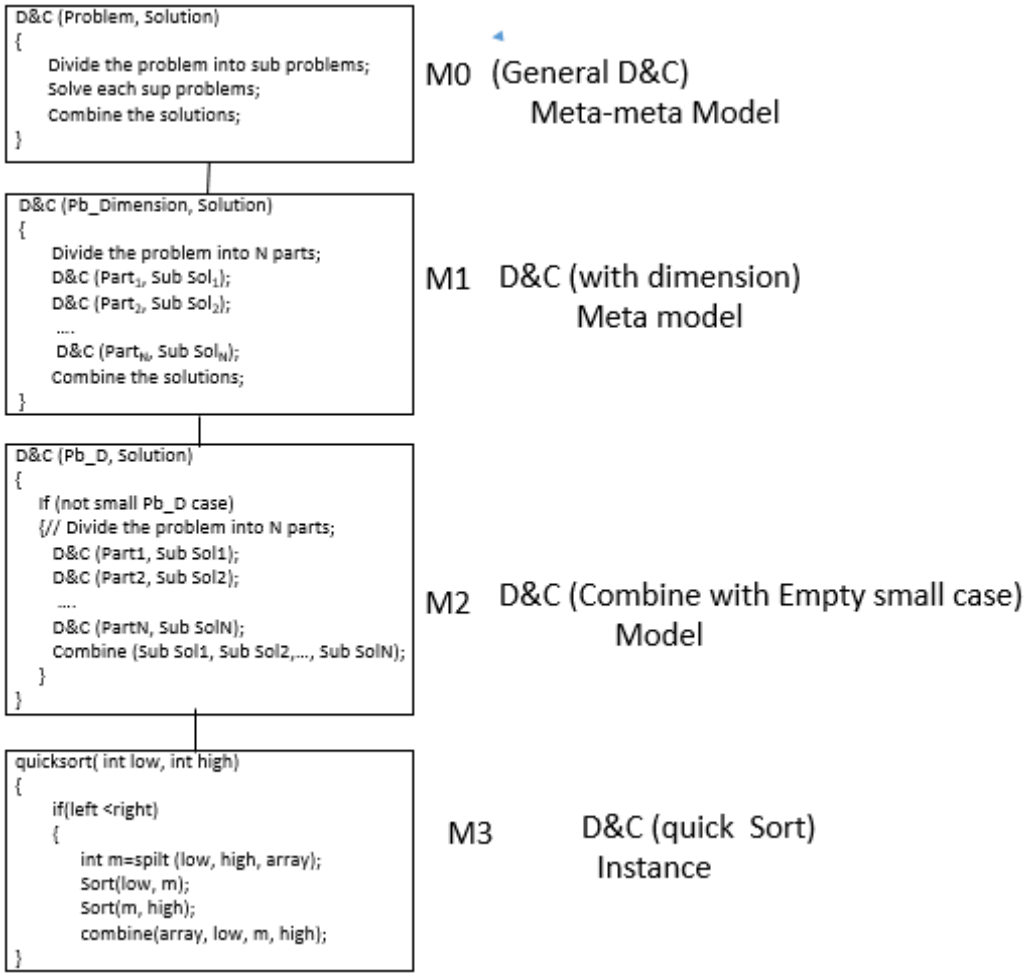



Figure 2-6. Meta Modelling divide and conquer.

Meta Modelling (MM) has been approached in several domains (Hao Wu, 2012; Gitze, 2005) in software engineering using Meta Modelling (MM) to create high abstraction model and presented systematic literature review of instance generation techniques for meta models.

The Meta Modelling (MM) is approached by some modelling language like Unified Model Language (UML) (Byrne, 2013) and Feature Diagrams (FD) (Kang, 1990).

## 2.4 Meta Modelling language

Meta Modelling language is a tool for supporting the meta model technique in representing the model. Some of these languages are Unified model language (UML) and Feature diagram (FD).

- **Feature Diagram**

Feature diagram (Kang, 1990) represents model as tree, this tree consists of the system's name at the top, and children for this system, whereas these children represent features.

Feature diagram has a huge relationship (Kang, 1990) between the features. We will use some relationship as in (Figure 2-7):

- Mandatory: this case means you must choose this child due to features.
- Optional: this case means you can choose this child or another child from features.
- Alternative: this case means you have multi children feature, you can choose only one feature from them.
- Or: this case means you have multi children feature, you can choose one or all feature from them.

Feature Diagram (FD) (Kang, 1990) used in software product line easily representing system.

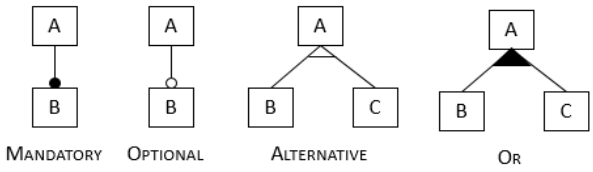



Figure 2-7. Relation between feature diagrams.

An example, the (Figure 2-8), presenting divide and conquer with combine case.

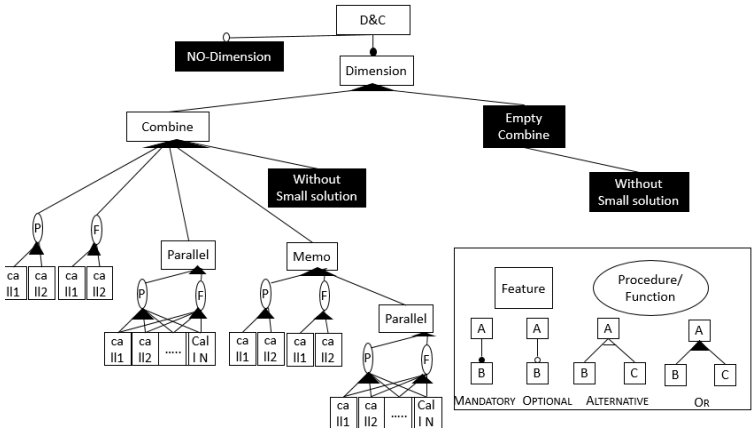



Figure 2-8. The model divide and conquer with combine case.

- **Unified Model Language (UML)**

Unified Model Language UML (Byrne, 2013) represents model and system in hierarchical form. There are many notation for representing model in UML that will support our study. Class diagram consists of: class name, attributes and operations. The attributes consist of global variable that will be used in the method, the operations consist of method from divide and conquer.

There are many relations (Byrne, 2013) between classes as in (Figure 2-9):

- Dependency: when a class needs data or information from another class.
- Association: when a class is connected with another class.
- Inherits: this relation connects father and son.
- Aggregation: this relation means multi class gives big relation.

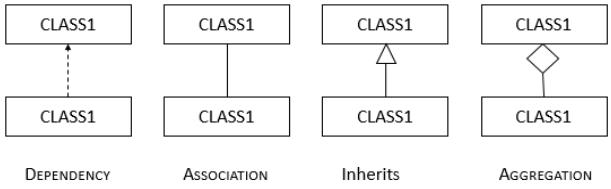



Figure 2-9. Relation between classes in UML.

An example, the (Figure 2-10), presenting divide and conquer by UML.

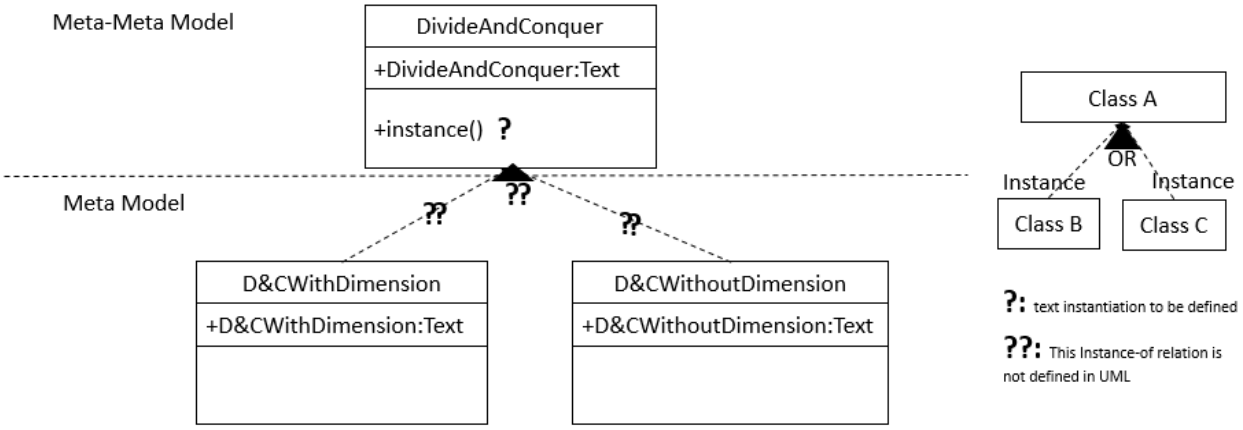



Figure 2-10. Divide and conquer model by UML.

## 2.5 Thesis Motivation

From the previously mentioned, based on meta model technique, supporting language like UML and FD and the research in divide and conquer method, this thesis is proposing general divide and conquer by Meta modelling.

This general model divide and conquer provides a methodology, that presents how to build meta model for divide and conquer and provides process to generate instance from general divide and conquer method.

# CHAPTER THREE: DIVIDE AND CONQUER VARIABLE CASES

## 3.1 Introduction

Divide and conquer may be used for solving broad classes of problem. In this chapter, several specific examples will be introduced for clarifying the problem. This example is among the thesis contributions. In fact they are developed for the thesis proposes.

## 3.2 Divide and conquer general pattern

The general case in divide and conquer works without dimension and with dimension.

### 3.2.1 Without dimension

This case means the problems will be divided irregularly into all programs steps (Figure 3-1).

```
D&C (Parameter, Solution)
{
   If (small problem case) then small solution;
   Else {// Divide the problem as current state
         D&C (Part1, Sub Sol1);
         D&C (Part2, Sub Sol2);
         ....
         D&C (PartN, Sub SolN);
         Combine (Sub Sol1, Sub Sol2,..., Sub SolN);
    }
}
```

Figure 3-1. Divide and conquer without dimension.

(Figure 3-1) presents divide and conquer algorithm without dimension. The first step consists of condition to stop divide problem, the second step divides problem into sub-problems as requirement, the third step solves each sub-problem, and the fourth step combines all sub-solutions.

Example search in tree (Figure 3-2):

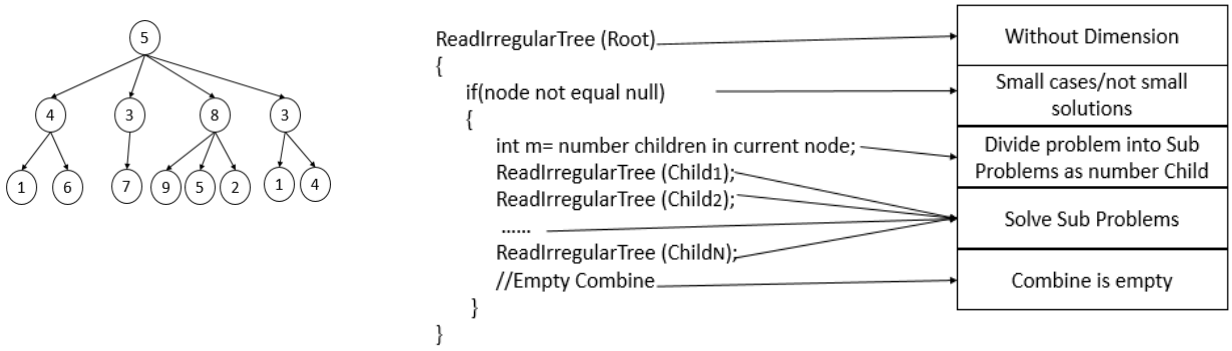



Figure 3-2. Read tree by divide and conquer without dimension.

This example is a specific case in divide and conquer method without dimension. The parameter in this case consists of Root that will read its children. The characteristics consist: firstly of small problem (node!= null) and without small solution. Secondly it consists of divide problem as child number. Thirdly it consists of sub problems (call same method) for new array size. The divide will stop if (node = null).

### 3.2.2 With dimension

This case means that the problems will be divided regularly into all programs steps (Figure 3-3).

```
D&C (Pb_D, Solution)
{
   If (small Pb_D case) then small solution;
   Else {// Divide the problem into N parts;
         D&C (Part1, Sub Sol1);
         D&C (Part2, Sub Sol2);
         ....
         D&C (PartN, Sub SolN);
         Combine (Sub Sol1, Sub Sol2,..., Sub SolN);
    }
}
```

Figure 3-3. Divide and conquer with dimension.

(Figure 3-3) presents divide and conquer algorithm with dimension (general case). The first step consists of condition to stop divide problem, the second step divides problem into N part in each algorithm circle, the third step solves each sub problem, and the fourth combines all sub solutions.

Example sum elements in Set (Figure 3-4):

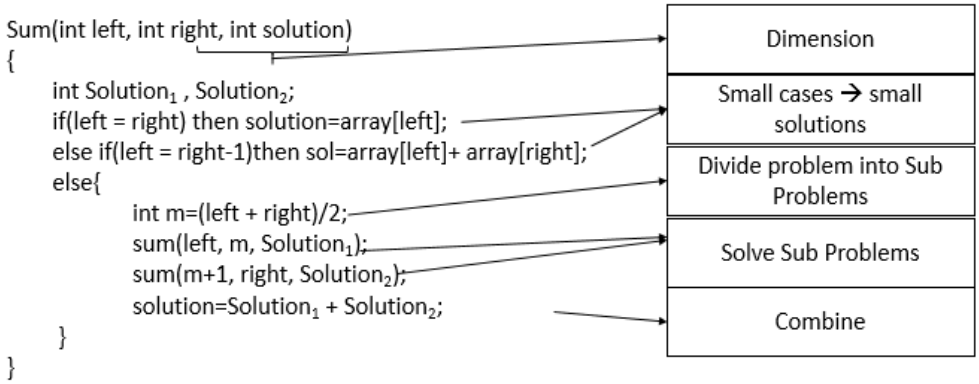



Figure 3-4. Sum by divide and conquer method.

This example is a general case in divide and conquer method with dimension. The parameters in this case consists of dimensions (left, right) which means that the problem will be divided into two parts, a solution means a result. The characteristics consist firstly of small solution (left=right or lift=right-1) and small solution (sol=array [left] or sol=array [left] +array [right]). Secondly it consists of divide problem. Thirdly it consists of sub problems (call same method sum) for new array size. The divide will stop to small solution. Fourthly it consists of combine, this characteristic means building solvation from previous one.

## 3.3 Divide and conquer method specializations

Many problems can use divide and conquer method, these problems differ from each other in parameters and characteristics. Some problems don't have small solution, while another do not have combine to solve problem etc.

### 3.3.1 Divide and conquer with empty small solution

Divide and conquer with empty small solution will be presented in (Figure 3-5).

```
D&C (Pb_D, Solution)
{
   If (not small Pb_D case)
   {// Divide the problem into N parts;
     D&C (Part1, Sub Sol1);
     D&C (Part2, Sub Sol2);
      ....
     D&C (PartN, Sub SolN);
     Combine (Sub Sol1, Sub Sol2,..., Sub SolN);
   }
}
```

Figure 3-5. Divide and conquer without small solution.

(Figure 3-5) presents divide and conquer algorithm with empty small solution. The first step, this algorithm does not have small solution but there is special condition to stop algorithm as (find element and equal index, Etc.), the second step divides problem into (N) part in each algorithm circle, the third step solves each sub-problem, and the fourth combines all sub solutions.

Example sorting algorithms may use divide and conquer method to sort items in a set. Sorting algorithm has many methods to sort item as (quick sort, bubble sort, merge sort etc.). A Rahmani presented sorting (Rahmani, 2010) model by UML as in (Figure 3-6).

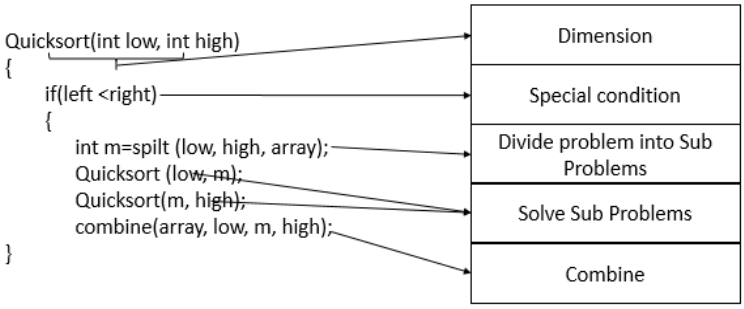



Figure 3-6. Sorting by divide and conquer method.

This example is a special case from general case, the parameters in this case are dimension (low, high) which means that the problem will be divided into two parts. The characteristics in this case differ slightly from those in the general case. Firstly there is not small solution (small case low< high), it has small case but it doesn't have small solution. Secondly it consists of divide problem, here the used method (split) to divide problem, and the split method differs from one method to another in sorting. Thirdly it consist of sub problems (call same method sort) for new array size, the divide will stop if (low>=high). Fourthly it consists of combine, this characteristic means building solvation by aggregation of elements in array.

### 3.3.2 Divide and conquer with empty combine

Divide and conquer with empty combine will present in (Figure 3-7).

```
D&C (Pb_D)
{
   If (small Pb_D case) then small solution;
   Else {// Divide the problem into N parts;
         D&C (Part1, Sub Sol1);
         D&C (Part2, Sub Sol2);
         ....
         D&C (PartN, Sub SolN);
         //Combine is empty;
    }
}
```

Figure 3-7. Divide and conquer empty combine.

(Figure 3-7) presents divide and conquer algorithm with empty combine. The first step consists of condition to stop divide problem, the second step divides problem into N part in each algorithm circle, the third step solves each sub problem, and there is not combine because the algorithm chooses one sub solution.

Example Read elements in Set (Figure 3-8):

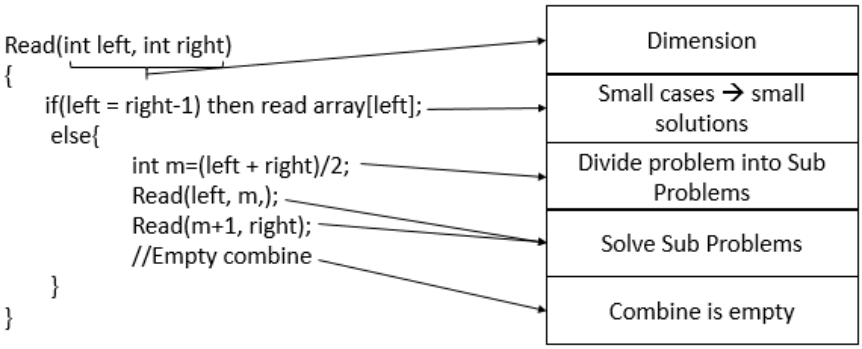



Figure 3-8. Read by divide and conquer method.

This example is a special case from the general case. the parameters in this case are dimension (low, high) which means that the problem will be divided into two parts, and there are many characteristics: Firstly of small case (left=right-1) and small solution (read array[left]). Secondly it consists of divide problem. Thirdly it consists of sub-problems (call same method Read) for new array size. Read method doesn't contain combine characteristic because the method does not interest in aggregation elements.

### 3.3.3 Divide and conquer empty combine with empty small solution

Divide and conquer empty combine without small solution in (Figure 3-9).

```
D&C (Pb_D)
{
   If (If (not small Pb_D case)
   {// Divide the problem into N parts;
     D&C (Part1, Sub Sol1);
     D&C (Part2, Sub Sol2);
      ....
     D&C (PartN, Sub SolN);
     //Combine is empty;
   }
}
```

Figure 3-9. Divide and conquer empty combine without small solution.

(Figure 3-9) presents divide and conquer algorithm with empty combine empty small solution. The first step, this algorithm does not have small solution but there is special condition to stop algorithm as (find element and equal index, Etc.), the second step divides problem into N part in each algorithm circle, the third step solves each sub problem, and there is not combine because the algorithm choose one sub solution.

Example Search elements in Set (Figure 3-10):

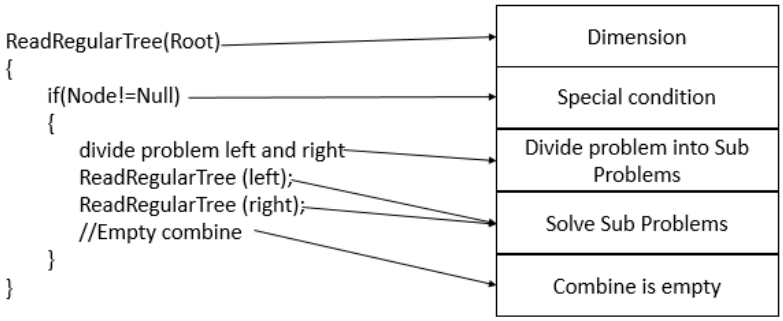



Figure 3-10. Read regular tree by divide and conquer algorithm.

This example is a special case from general case. There are one parameters (Root) which means dividing the problem into two dimension, and there are many characteristics: Firstly of small case (Node!=Null) without small solution. Secondly it consists of divide problem. Thirdly it consists of sub problems (call method Read) to read the children. Read method does not contain combine characteristic because the method needs read element.

### 3.3.4 Divide and conquer with parallelism

The presence of multi process has been driven to think about the parallel case by divide and conquer method (Lopez-Ortiz, 2014) in (Figure 3-11).

```
D&C (Pb_D, Solution)
{
   If (small Pb_D case) then small solution;
   Else {// Divide the problem into N parts;
      Do in Parallel {
         D&C (Part1, Sub Sol1);
         D&C (Part2, Sub Sol2);
         ....
         D&C (PartN, Sub SolN);
         Combine (Sub Sol1, Sub Sol2,..,Sub SolN);
      }
   }
}
```

Figure 3-11. Divide and conquer with parallel.

(Figure 3-11) presents divide and conquer algorithm with parallel. The first step consists of condition to stop divide problem, the second step divides problem into (N) part in each algorithm circle, the third step solves each sub-problem and all sub-problem works in same time on separately process, and the fourth combines all sub solutions.

Example Sum elements in Set with Parallel (Figure 3-12):

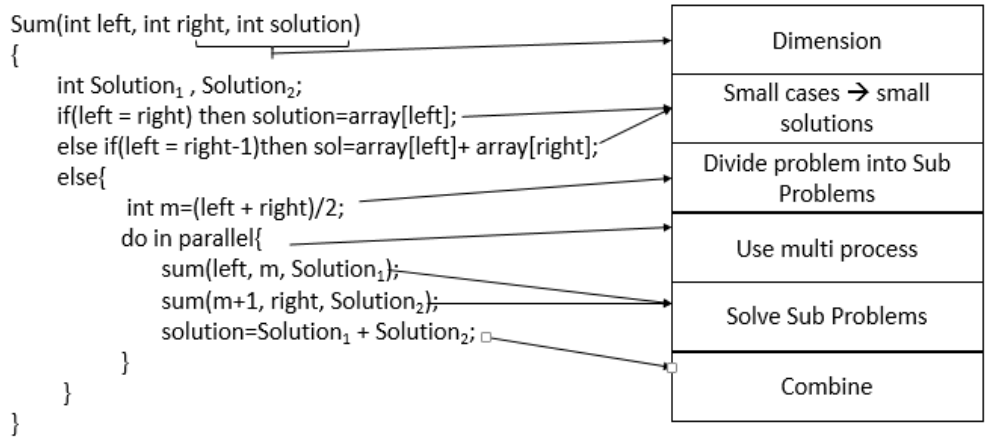



Figure 3-12. Sum in parallel by divide and conquer method.

The new in example, parallel was used by divide and conquer method. When using parallel the time will be faster for problems solution by divide and conquer. Divide and conquer can be used in parallel with all previous cases.

### 3.3.5 Divide and conquer with memorization

Sometimes divide and conquer uses to solve the same problem, instead of dividing the problem into smallest case, memorization can be used to save the result. (Figure 3-13).

```
Memory A
D&C (Pb_D, Solution)
{
   If (current case in A) then Solution ← Solution from A;
   Else {
        If (small Pb_D case) then save small solution in A
         Else {// Divide the problem into N parts;
                D&C (Part1, Sub Sol1);
                 D&C (Part2, Sub Sol2);
                 ....
                 D&C (PartN, Sub SolN);
                Combine and save in A (Sub Sol1, Sub Sol2,...,Sub SolN);
         }
   }
}
```

Figure 3-13.Divide and conquer with memorization.

(Figure 3-11) presents divide and conquer algorithm with memorization. The first step searches in memory if find solution, the algorithm will stop. The second step consists of condition to stop divide problem, the third step divides problem into N part in each algorithm

circle, the fourth step solves each sub problem, and the fifth combine all sub solutions and save it in memory.

Example Factorial number with memorization (Figure 3-14):

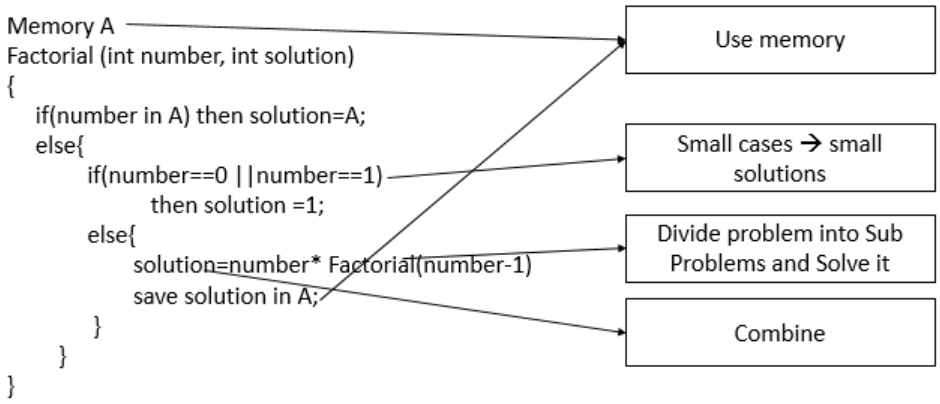



Figure 3-14. Factorial by divide and conquer with memorization.

Usually, memorization is used with static number because in every use the method will repeat the same step. (Figure 3-14) using memory to save the result. The parameters (number) the value is used to find factorial and (solution) to return result. This method will search in memory to find the result stop method otherwise will divide problem and calculate the factorial.

Divide and conquer can use memorization with all previous cases.

# CHAPTER FOUR: A VARIABILITY META MODEL IN DIVIDE AND CONQUER METHOD

## 4.1 Introduction

This chapter presents the thesis contribution to the divide and conquer general methodology, meta modelling by Feature Diagram (FD) and unified model language (UML), and instance generating process.

## 4.2 A methodology for designing algorithms variability by divide and conquer

In the following, a divide and conquer meta modelling methodology design will be introduced, the methodology defines the main activities and their coordination producing a divide and conquer algorithm specific to a given problem. The main activities are: divide and conquer meta modelling, problem specific algorithm meta model selection request, Problem specific algorithm meta model instantiation, problem specific algorithm variability parameters fixing, and Problem specific algorithm.

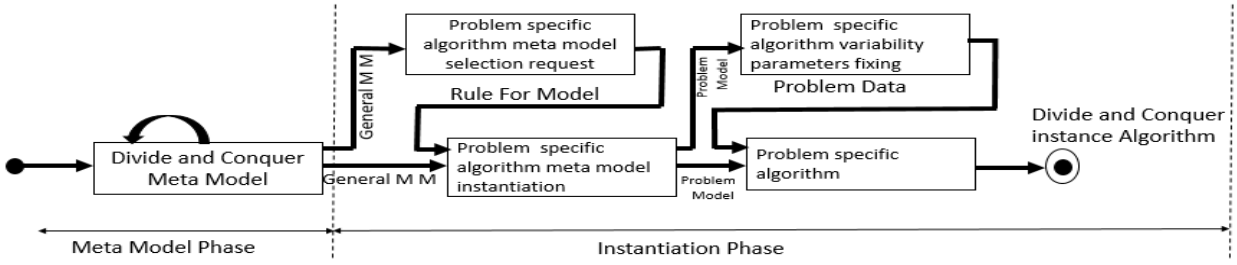



Figure 4-1. Divide and Conquer Meta Modeling methodology, using Data flow Diagram.

This methodology has scenario in appendix to explain mechanism of action this methodology by example.

The first step is providing a general meat modelling for divide and conquer. The second step consists of two phases: (1) problem meta model selection request for determining a user specific for a given problem. (2)Meta Modelling instantiation that produces a specific meta modelling for the given problem. The third step produces divide and conquer instance algorithm by filling the variable parts in the selected metamodel for the fixed problem, these parameters were provided by the user. In the following, details of each methodology will be presented.

## 4.3 Divide and Conquer Algorithms Variability Meta Modelling Phase

The D&C variability meta model is shown in (Figure 4-2). The first layer consists of meta-meta model. The second layer consists of meta model which presents divide and conquer having dimension and divide and conquer without dimension. The third layer consists of model which presents (Combine, Combine with empty small case, Empty Combine and Empty Combine with Empty small case). The fourth layer consists of instance presenting the algorithm model from layer three (example sum, sort, read, BFS, DFS and etc. (Figure 4-2).

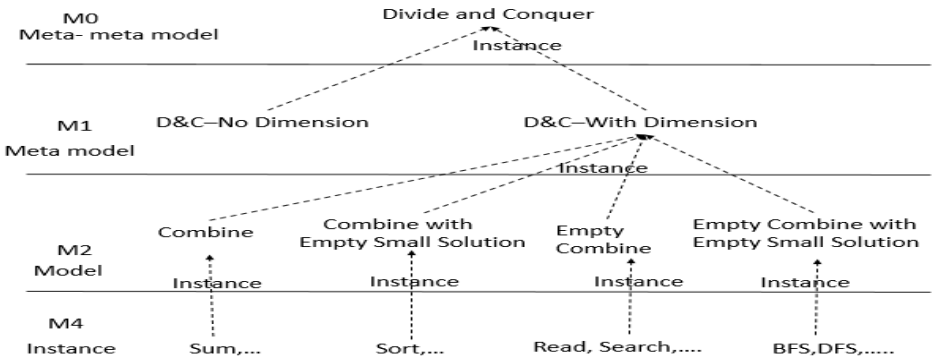
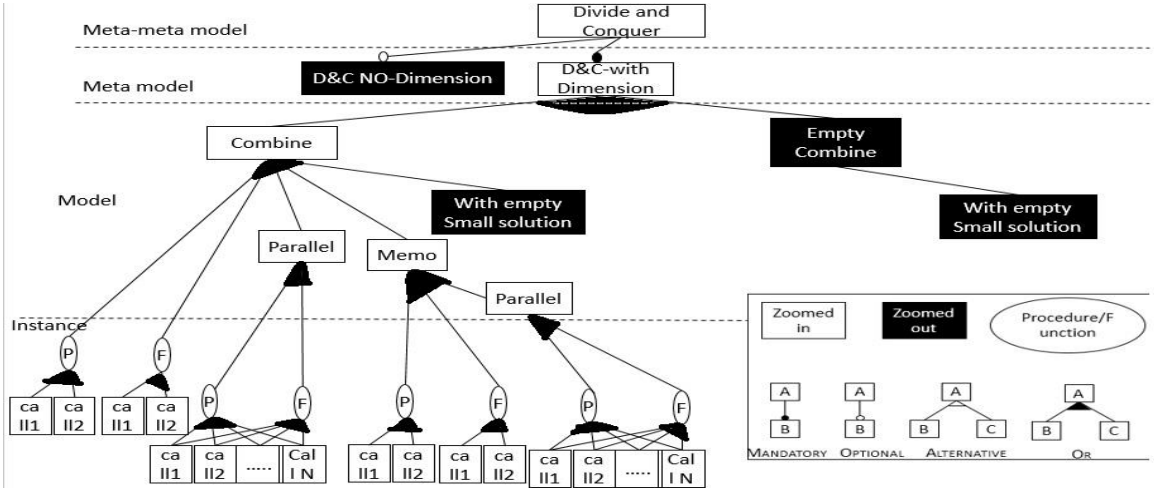



Figure 4-2. Divide and conquer by meta model technique.

### 4.3.1 Divide and conquer meta modelling by Feature Diagram

In this section, The above D&C meta model (Figure 4-2), will be summary formalized by feature diagram notations (Figure 4-3-A, Figure 4-3-B, Figure 4-3-C, Figure 4-3-D). D&C method variations modelling study for problems without dimension is out of scope of this thesis.



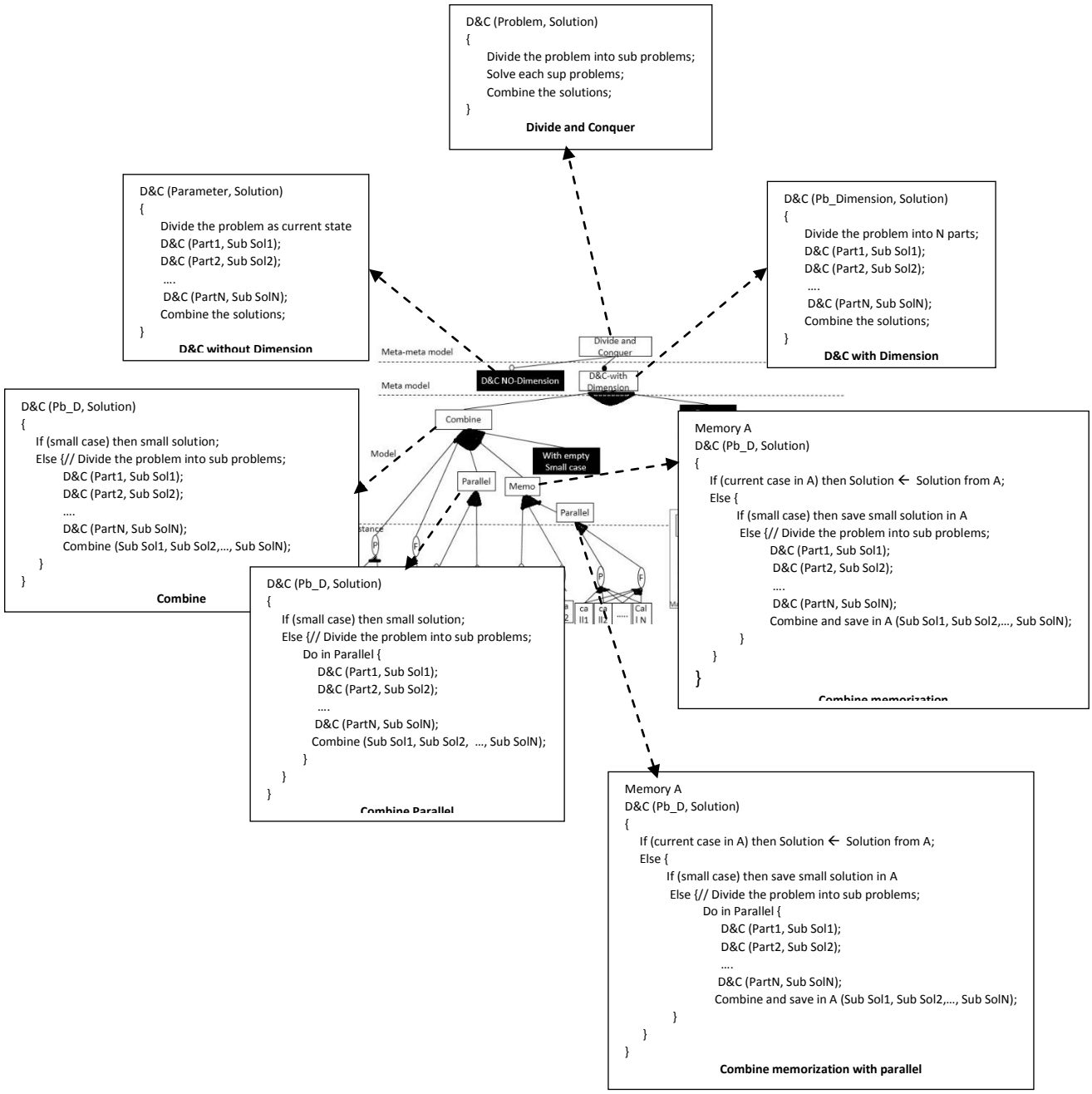



Figure 4-3-A. Divide and conquer method with combine and Text.

(Figure 4-3-A) presents a Combine variation FD of D&C method. Combine came from dimension parameters. It consists of leaves (procedure or function) and many variations features (with empty small case, parallel, memorization, memorization with parallel). Each variation ends by leaves (procedure or function) with the number of call. The (Figure 4-3-A) shows the Combine variations textual meta model.

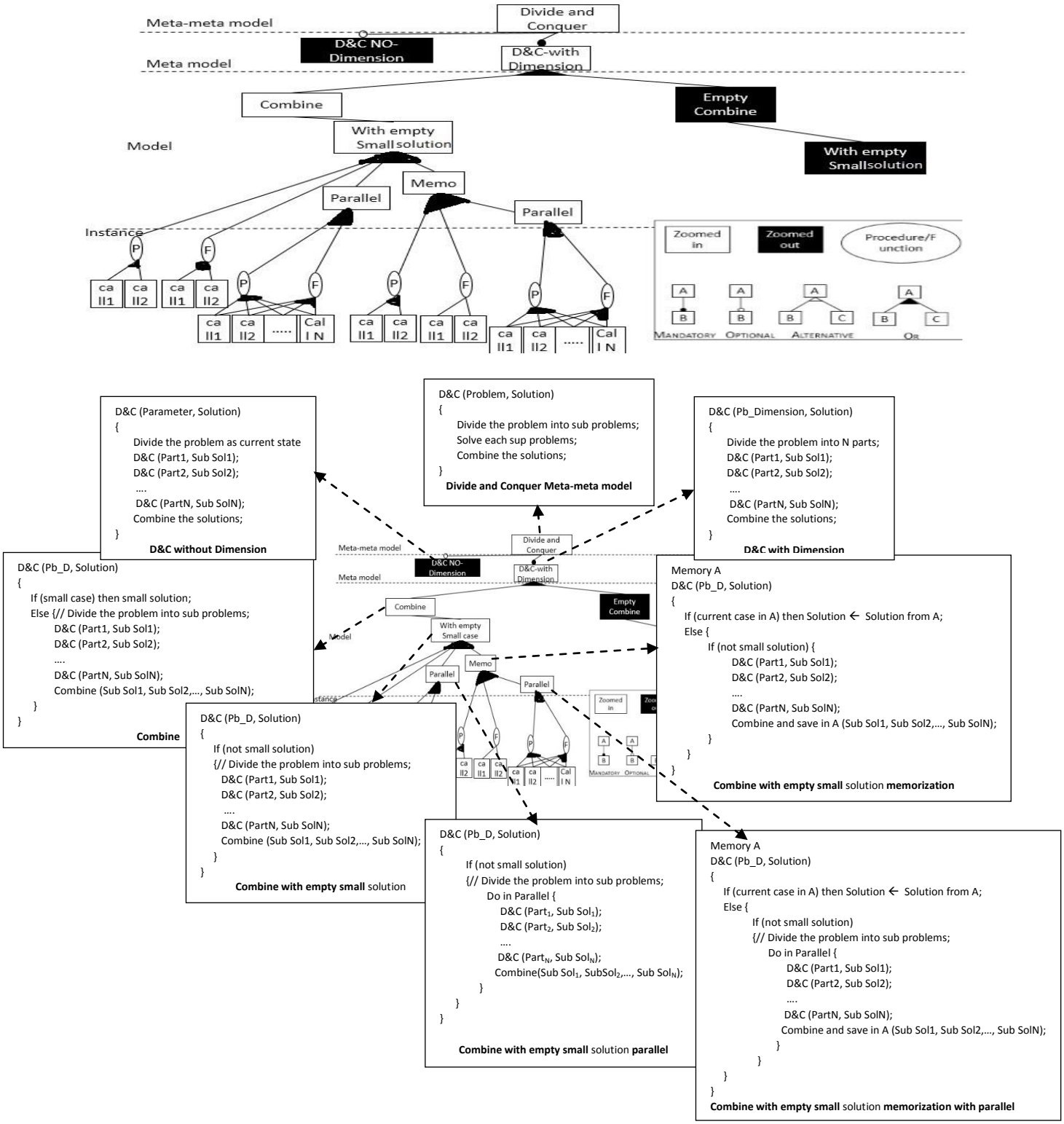



Figure 4-3-B. Divide and conquer method with combine with empty small solution and Text.

(Figure 4-3-B) shows combine with empty small solution variation FD of D&C method. With empty small solution coming from combine, and combine coming from dimension

parameters. It consists of leaves (procedure or function) and many variations features (parallel, memorization, memorization with parallel). Each variation ends by leaves (procedure or function) with the number of call. The (Figure 4-3-B) shows the combine with empty small solution variations textual meta model.

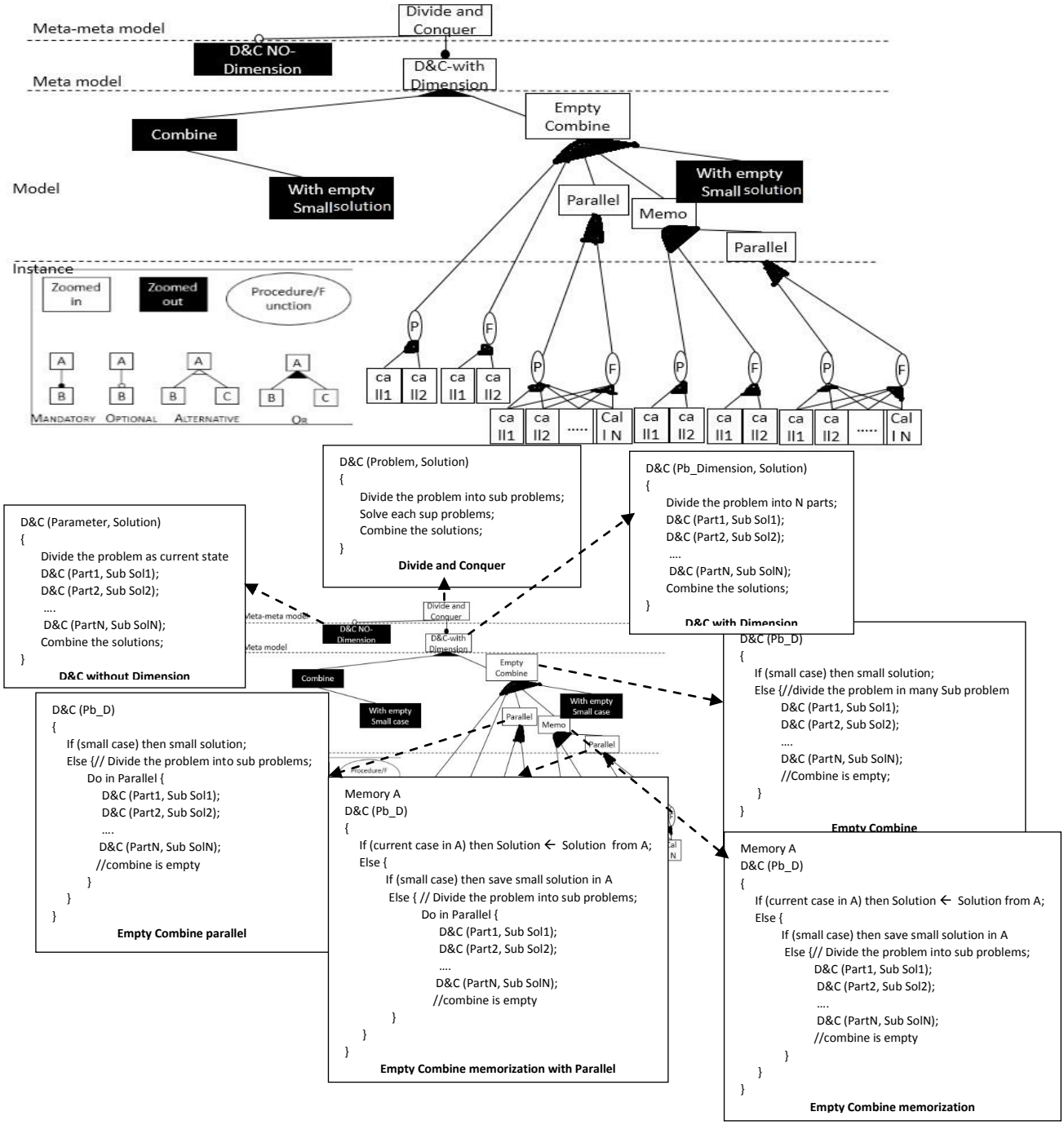
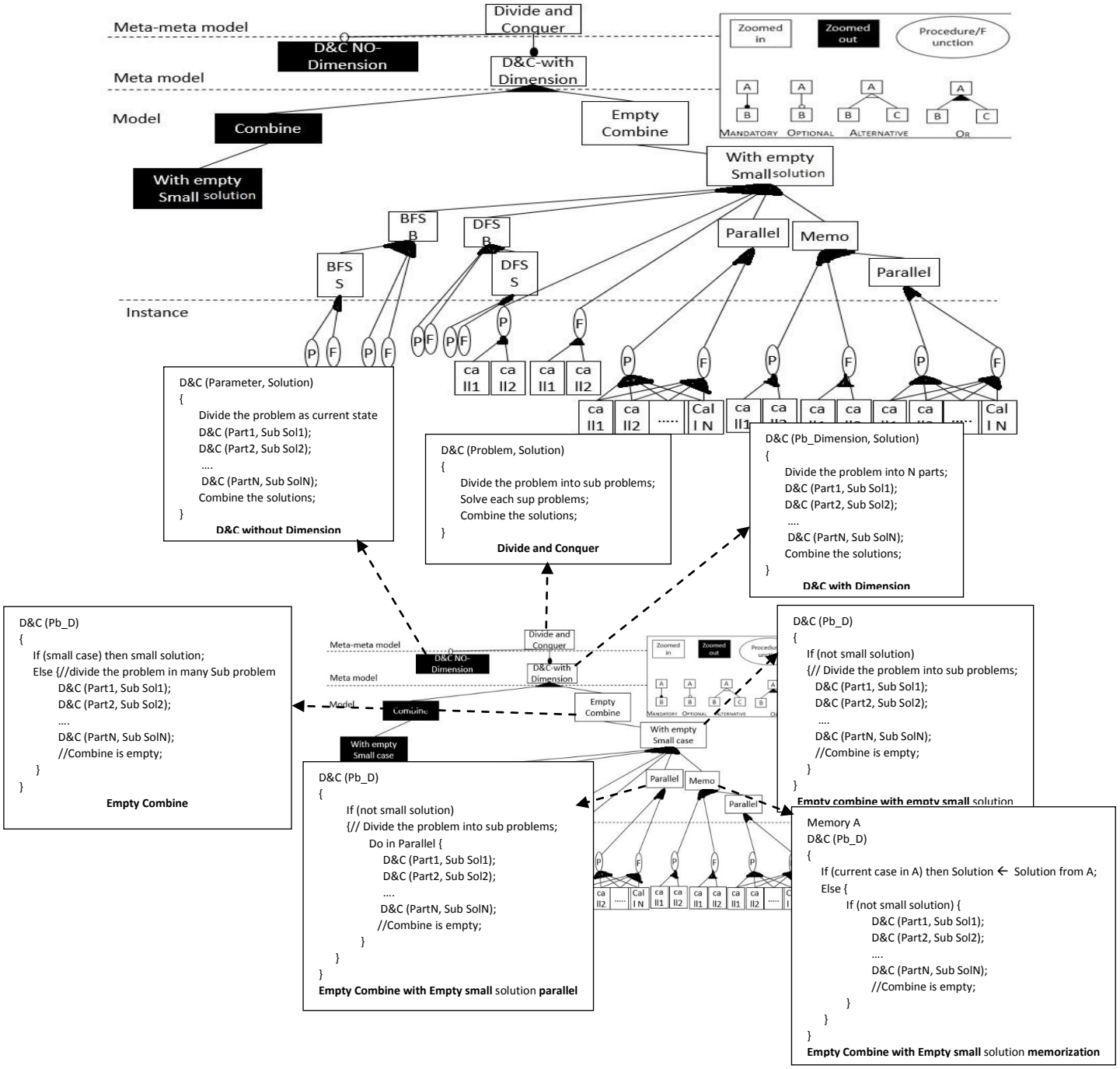



Figure 4-3-C. Divide and conquer method with empty combine and Text.

(Figure 4-3-C) presents empty combine case that comes from dimension parameters. It consists of leaves (procedure or function) and many variations features (with empty small case, parallel, memorization, memorization with parallel). Each variation ends by leaves (procedure or function) with the number of call. The (Figure 4-3-C) shows the Combine variations textual meta model.



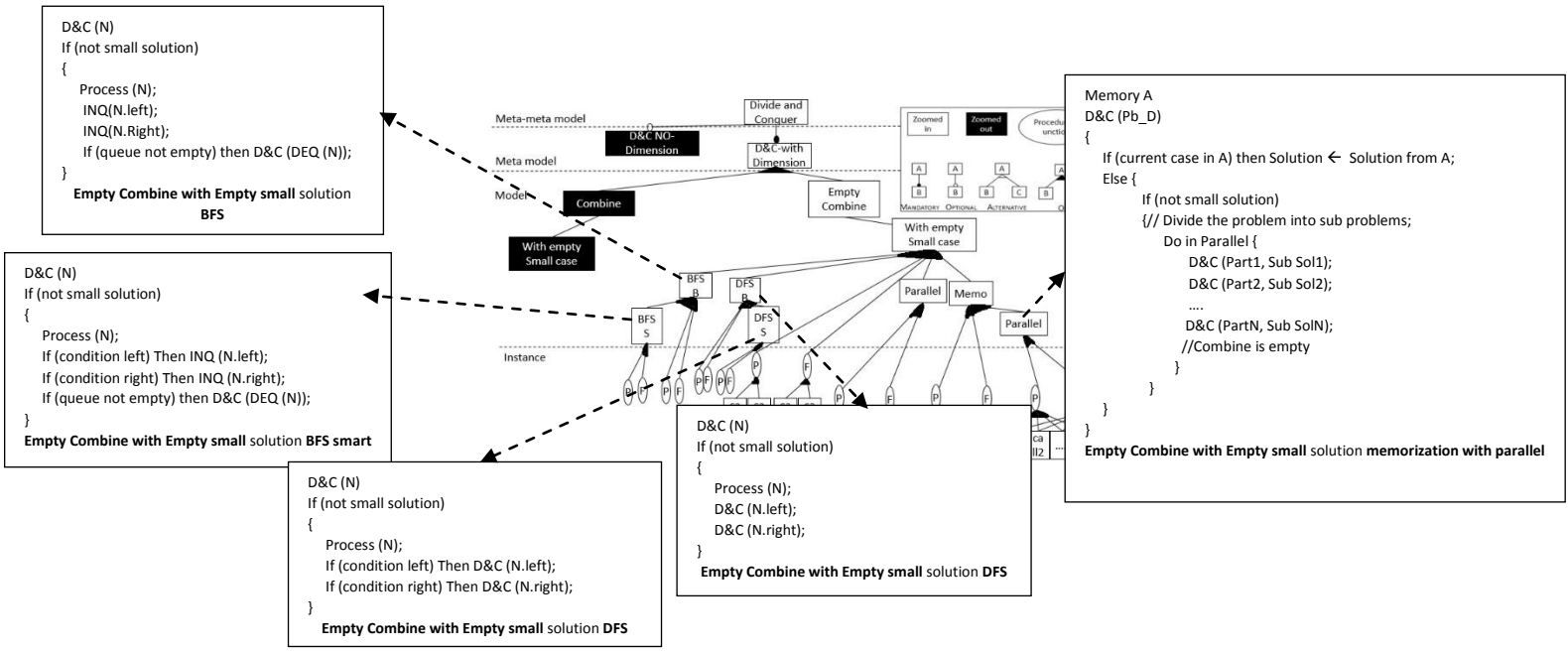



Figure 4-3-D. Divide and conquer method with empty combine with empty small solution and Text.

(Figure 4-3-D) presents empty combine with empty small solution. It consists of leaves (procedure or function) and many variations features (parallel, memorization, memorization with parallel and tree). Each variation ends by leaves (procedure or function) with the number of call. The tree feature has two children DFS B (depth first search blind) and BFS B (breadth first search blind). Each child has three features, two leaf (procedure and function) and child contain S (smart) then this feature arrive to leaf (procedure and function). The (Figure 4-3-D) shows the combine with empty small solution variations textual meta model.

### 4.3.2 Divide and conquer meta modelling by UML

This section tries to present the above D&C meta model (Figure 4-2) with UML class diagram notations.

The primary class in meta-meta model layer is DivideAndConquer class, this class has attribute (DivideAndConquer as Text) and operation (instance). The next layer is of two classes, the first class D&CWithDimension consists of attribute (D&CWithDimension as Text). The

second class D&CWithoutDimension consists of attribute (D&CWithoutDimension as Text). The relation between classes is instance.

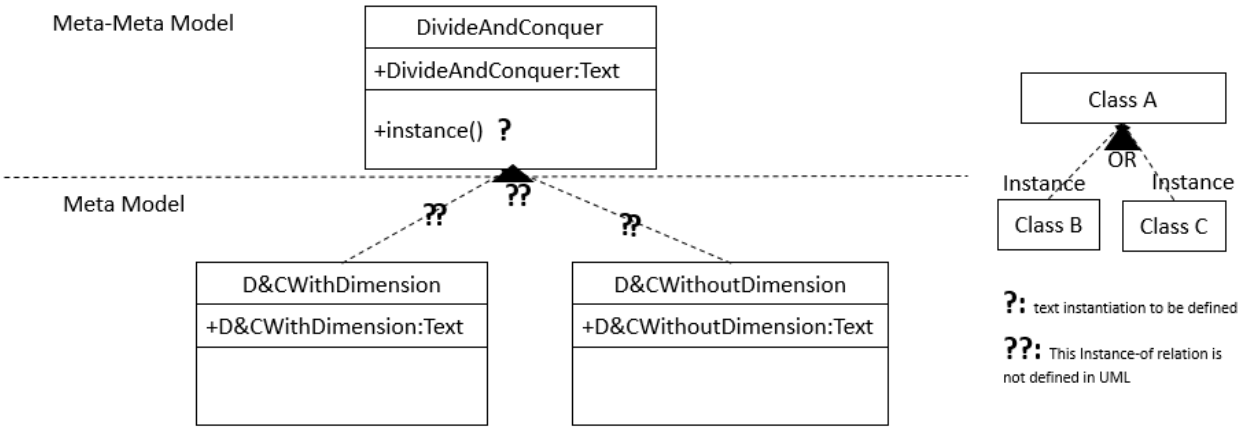



Figure 4-4. Divide and conquer model by UML.

The idea in (Figure 4-2), is that the bottom layer takes instance from above layers and changes the text at the above layers to arrive at a new instance divide and conquer algorithm as in the methodology in (Figure 4-1).

UML is suitable for operational problems but not for textual ones. So, it does not support the instantiation concept nor the text concepts. The following problems are clearly stated:

- There is no textual instance relation in UML class diagram notation.
- In class diagram all attributes and operations in primary class will be contained in children class, this machine is not needed in divide and conquer meta modeling, which is a text modeling.
- The attribute and operation do not change from one class to another in D&C class diagram notation. So each children class will contain new text as attribute, derived from the ancestor text by a special instantiation relation.

## 4.4 Problem Specific Algorithm Instantiation Phase

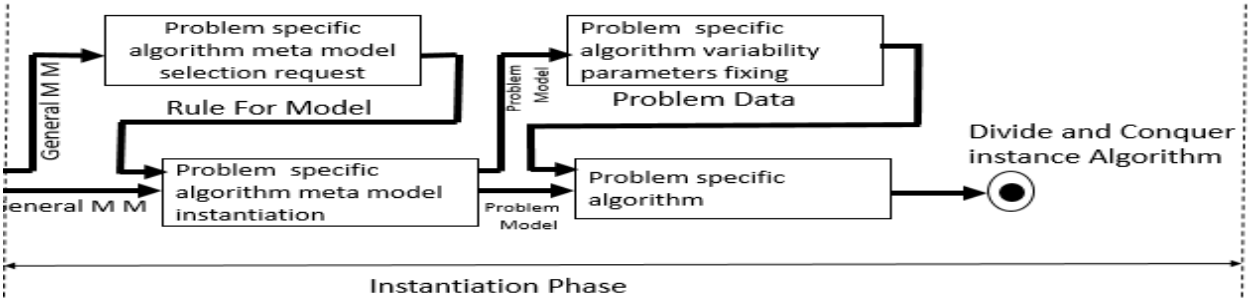



Figure 4-5. Instantiation phase, using data flow diagram notations.

Now there is the need for a technique to generate instance from general meta model, the process will be divided into many steps. The first step will depend on entering sentence describing specific problem from user or from example and process it. The second step generate a problem meta model (schema). The third step will modify the problem meta model by the user. The fourth part will product a D&C instance algorithm for the fixed problem.

### 4.4.1 Problem specific algorithm meta model selection request

The user will enter a sentence that describes the problem, this sentence must be processed to convert it to a selection rule. If the sentence does not produce specific rule, the user will enter an example, this example must be processed to convert it to a selection rule. If there is no specific rule, the system will produce a default rule (Figure 4-6).

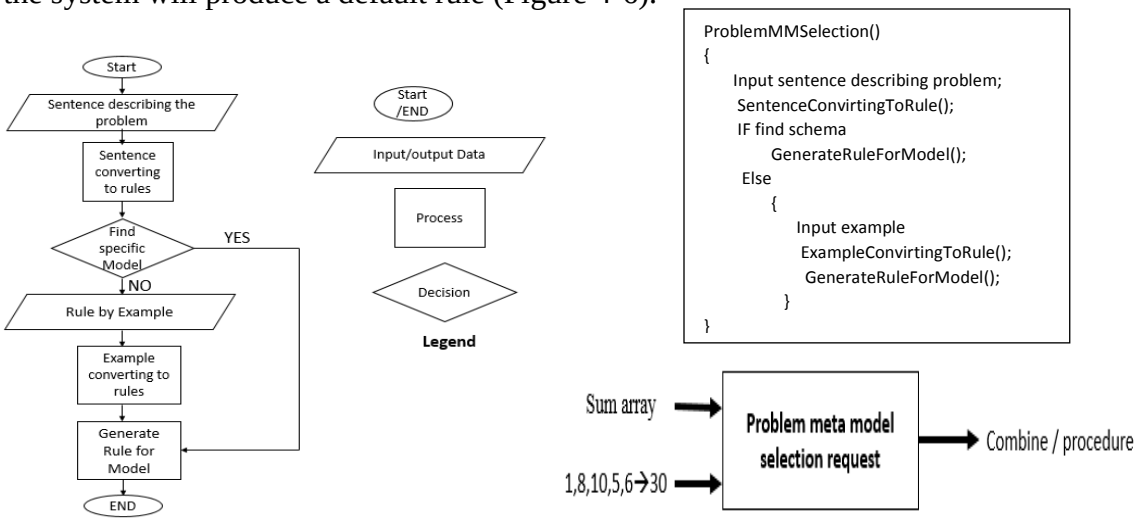



Figure 4-6. Problem specific algorithm meta model selection request and process, using flow charts and Example.

(Figure 4-6) presents example of problem specific algorithm meta model selection request. The user will give a sentence (sum array) or an example (1, 8, 10, 5, 6 →30), the process will generate the suitable rule (combine/procedure).

- **Sentence describing the problem:**

The following BNF rules specify the used language for describing the specific requirements of the problem at hand:

<Sentence>:= <key words>
<Key words>:= (<key word>) +
< Key word > ::=< default>
|array
|set
|list
|sort
|Search
| (Depth || top down || dfs) ∨ (| breadth || level || bfs)
|smart
| (Sum || total || +)
| (multiply || *)
|(Less || lower || min)
| (Great || max || higher)
|read
|write
|Insert
|delete
|factorial
|static data || use again || memory
|parallel ||multi thread || multi process || multi cpu
|procedure ∨ function

<Default>:=procedure
|two call
||: the words same meaning; V: the words do not use together

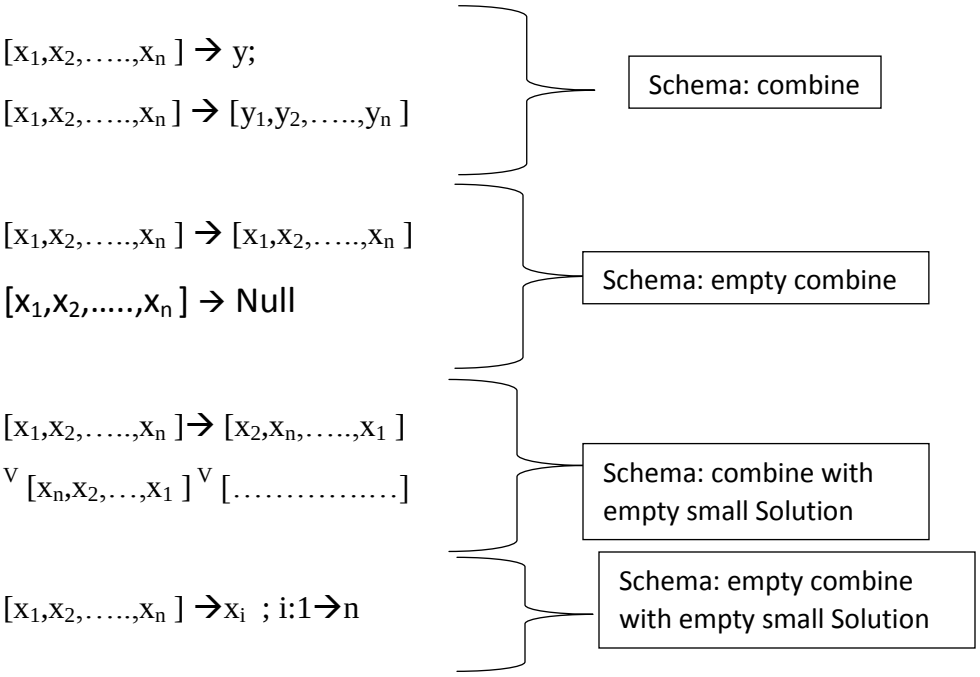
- **Example sentence:**
  - Sum array, Multiply list, Insert array, Read set, Sum set parallel, Write set parallel, function, Factorial static, Search array, Search set parallel function, BFS smart, Sort list function, Sort set parallel,…….
- **Sentence converting to rule :**
  - Sum array V Sum set V Sum list →combine.

- Total array V Total set V Total list → combine.
- + array V + set V + list → combine.
- Multiple array V Multiple set V Multiple list → combine.
- * array V * set V * list → combine.
- Less array V Less set V Less list → combine.
- Lower array V Lower set V Lower list → combine.
- Min array V Min set V Min list → combine.
- Great array V Great set V Great list → combine.
- Max array V Max set V Max list → combine.
- Higher array V Higher set V Higher list → combine.
- Factorial array V Factorial set V Factorial list → combine.
- Sort array V Sort set V Sort list → combine / empty small solution.
- Read array V Read set V Read list → empty combine.
- Write array V Write set V Write list → empty combine.
- Insert array V Insert set V Insert list → empty combine.
- Delete array V Delete set V Delete list → empty combine.
- Search array V Search set V Search list → empty combine.
- BFS V breadth V level → empty combine / empty small solution / BFS.
- BFS smart V breadth smart V level smart → empty combine / empty small solution / BFS S.
- DFS V depth V top down → empty combine / empty small solution / DFS.
- DFS smart V depth smart V top down smart → empty combine / empty small solution / DFS S.
- Static data V use again V memory → memorization.
- Parallel V multi thread V multi process V multi CPU → parallel.
- Procedure → Procedure.
- Function → Function.
- Default rule → combine / procedure

- **Example of rules:**
  - Sum array →[combine / procedure].
  - Multiply list → [combine / procedure].
  - Insert element to array →[empty combine / procedure].
  - Read set →[empty combine / procedure].
  - Sum set parallel →[combine / parallel / procedure].
  - Write set parallel function → [empty combine / parallel / function].
  - Factorial static data →[combine / memorization / procedure].
  - Search array →[empty combine / empty small solution / procedure].
  - BFS S function → [empty combine / empty small solution / parallel / function].
  - Sort list function → [combine / empty small solution / function].
  - Sort set parallel → [combine / empty small solution / procedure].

- **Rule by example and converting example to rule:**

If the process fails to select instance from sentence, the user will enter example the process will analysis example by rule:




- **Example of rules:**
  - 1, 9, 16, 81,121→1, 3, 4,9,11→ combine.
  - 1,12,15,4,3,2→37→ combine.
  - 11, 20,21,33,44→11, 20,21,33,44→ empty combine.
  - 1,8,7,6,5→Null→empty combine
  - 12, 11,0,1,4→4,1,0,11,12→ combine / empty small solution.
  - 100, 14,15,12,50→14→: empty combine / empty small solution

### 4.4.2 Problem specific algorithm meta model instantiation

Given a selection rule and the general meta model, the instantiation process will produce the instance candidate for the given problem (Problem model).If specific instance is found, the process will generate it. Otherwise the process will generate a general model (Figure 4-7).

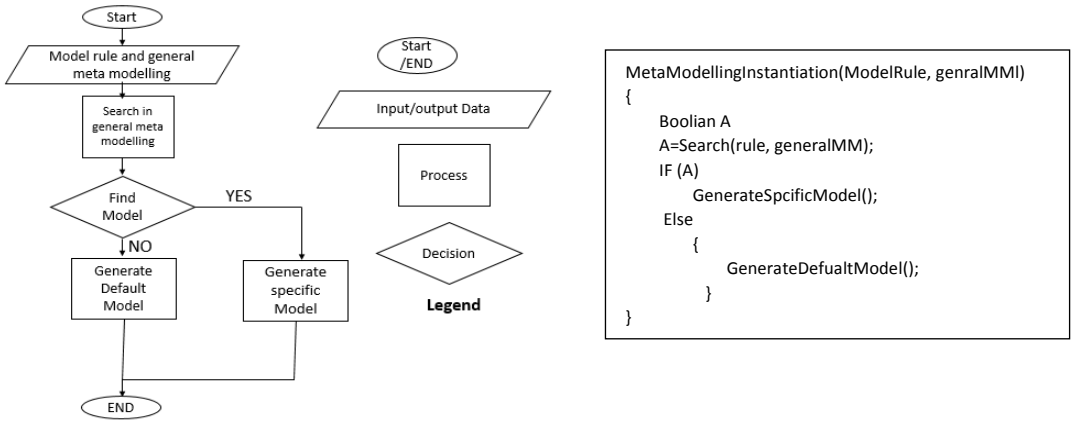



Figure 4-7. Problem specific algorithm meta model instantiation and process, using flow charts.

- **Example:**

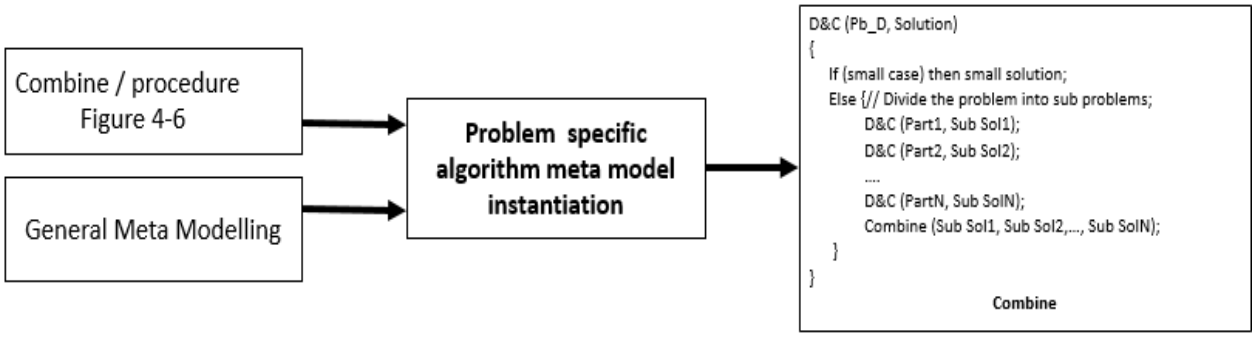



Figure 4-8. Problem specific algorithm meta model instantiation Example.

(Figure 4-8) presents example of problem specific algorithm meta model instantiation. The process will take selection rule (combine/procedure, obtained in Figure 4-6) and general meta model, the process will generate the suitable model (combine, obtained in Figure 4-8).

### 4.4.3 Problem specific algorithm variability parameters fixing

This process identifies the variable fields which are the instance parameters for which the user should provide values (Figure 4-9).

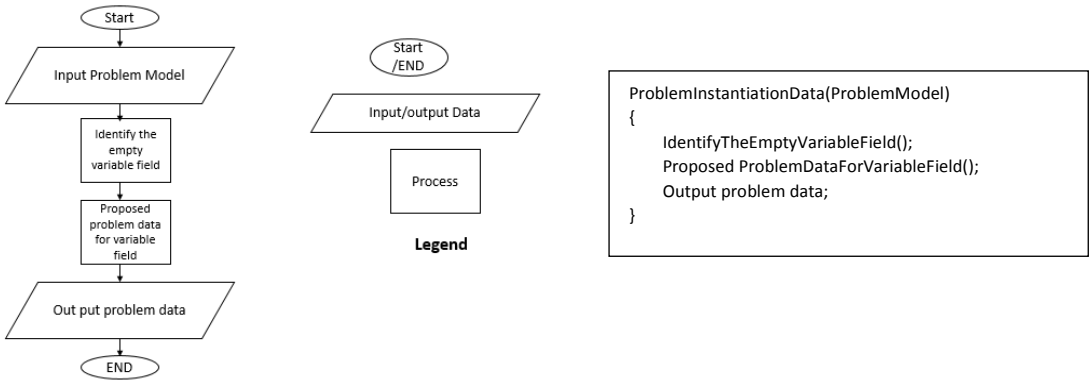



Figure 4-9. Problem specific algorithm variability parameters fixing and process, using flow charts.

- **Example:**

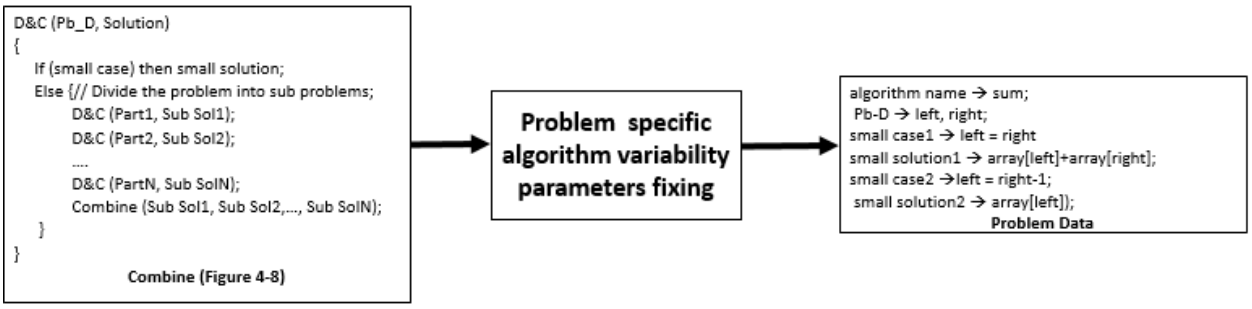



Figure 4-10. Problem specific algorithm variability parameters fixing Example.

(Figure 4-10) presents example of problem specific algorithm variability parameters fixing. The user will identify the fixed parameters (problem data) for selection model (combine, obtained in Figure 4-8), the process will take the variable fields from user:

algorithm name→sum;

Pb-D→left, right;

small case1 → left = right

small solution1 → array[left]+array[right];

small case2 →left = right-1;

small solution2 → array[left]);

After the user provides the value, the problem specific algorithm variability parameters fixing process will send the value (in Figure 4-10) to problem specific algorithm process.

### 4.4.4 Problem specific algorithm

This process fills the variable fields of an instance by the user fixed parameters textual values (Figure 4-11).

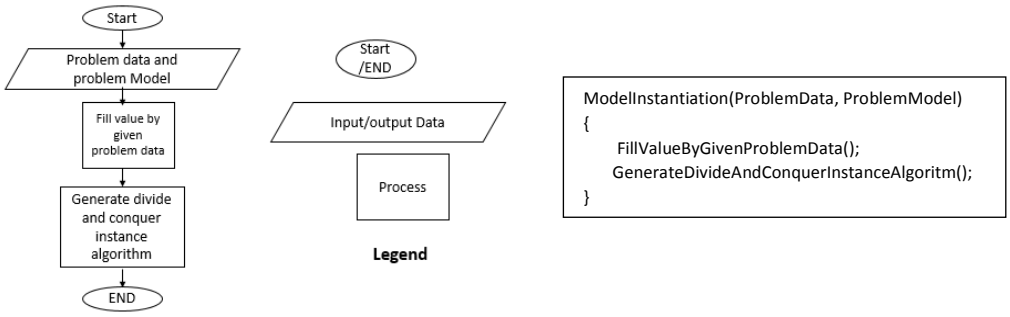



Figure 4-11. Problem specific algorithm and process, using flow charts.

- **Example:**

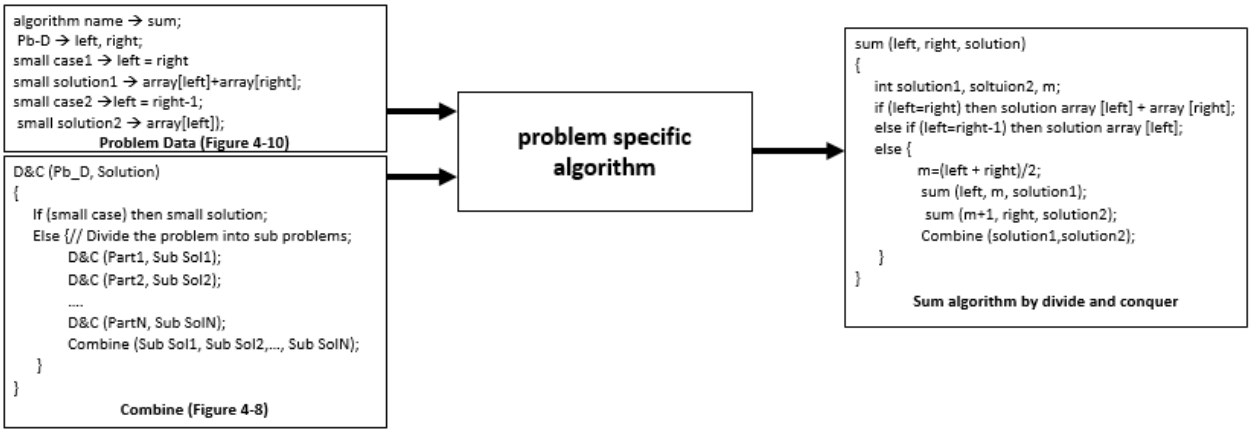



Figure 4-12 Problem specific algorithm Example

(Figure 4-12) presents example of problem specific algorithm. The process will fill the fixed parameters (obtained in Figure 4-10):

algorithm name→sum;

Pb-D→left, right;

small case1 → left = right

small solution1 → array[left]+array[right];

small case2 →left = right-1;

small solution2 → array[left]);

After filling the parameters of an instance (combine, obtained in Figure 4-8) then it will produce sum algorithm by divide and conquer (obtained in Figure 4-12).

## 4.5 Discussion

This thesis has designed a methodology for D&C method. This methodology is based on meta modelling D&C variability and on an instantiation process according to problem soecific requirements. The formalization of this methodology by meta modelling well known languages (UML and FD) has proved the non-suitability of UML to Text architectures modelling, whereas the FD was stated very adapted to.

# CHAPTER FIVE: IMPLEMENTATION ISSUES, APPLICATION AREAS AND PERSPECTIVE

## 5.1 Introduction

This chapter discusses the following views of the thesis contribution: The first point deals with implementation issues, the second point deals with the application areas of that contributions, the third point deals with evaluating the contribution by comparing it with relevant works, and finally it presents a conclusion and future possible works.

## 5.2 Implementation issues

No specific requirements in programmed environment and skills are required.

## 5.3 Application areas

Divide and conquer method will be strengthened by using meta modelling concept, it will add higher general abstraction and generate instance easily.

The proposed methodology may be used in all dimensioned problem solving, having variation parameters from one to another.

## 5.4 Evaluation

This section starts by comparing some relevant related works with this thesis contribution, based on some identified relevant criteria. It ends by evaluation UML and DF regarding their suitability to variable textual architectures meta modelling. Bellow supporting concept example in Table 5-2, and the Symbol using: × this means not supported.

Table 5-1. Divide and conquer meta modelling concept

| Concept | Example of supporting works | Thesis concept supporting |
|---|---|---|
| Divide and conquer meta model | × | Apply meta model concept to divide and conquer method. |
| Divide and conquer model | (Francés, 1998) | Represented divide and conquer cases by Feature Model. |
| Divide and conquer instance schema | (Rahmani, 2010). | A schema product from divide and conquer model. |
| Divide and conquer instance algorithm | × | There is a technique to generate divide and conquer instance algorithm. |

***Comparison with similar works:*** through reading the literature (Francés, 1998; Rahmani, 2010), there are many evaluation criteria:

1. Methodology. Actually, there is no methodology for divide and conquer meta modelling. This thesis proposes a methodology for it.
2. General Abstraction. There is a fixed and operational design pattern in (Francés, 1998) and Sorting Algorithms operational framework model in (Rahmani, 2010). This thesis proposes a general and variable textual metamodel covering broad classes of problems
3. Modelling Languages. The current works, which are limited to operational Divide & Conquer model (one abstraction level) uses UML (Object-Oriented operational modelling language) which is very suitable for this abstraction modelling level. This thesis proposes a meta model with several abstraction levels, each one deals with a variability parameters in Divide and Conquer. This kind of variability requires text-based meta modelling languages rather than operational languages. So, the use of FDL (which is abstractions multi layered and text-based) revealed to be very suitable, whereas UML was ineffective.
4. Instantiation Process. No, instantiation Processes were proposed in the current research works in this domain, this thesis proposes, as part of its Divide & Conquer metamodeling methodology, a process for this important task, with all its needed mechanisms.

Bellow a comparison represented in Table 5-2, and the Symbol using: ✓✓ means complete support, ✓ means partial support, and ✗ means not support.

Table 5-2. Comparison between related work and thesis contribution

| Concept | Francés 1998 | Rahmani 2010 | thesis contribution |
|---|---|---|---|
| Methodology | ✗ | ✗ | ✓✓ |
| Abstraction general | ✓ | ✓ | ✓✓ |
| Feature diagram | ✗ | ✗ | ✓✓ |

| | | | |
|---|---|---|---|
| UML | ✓ | ✓ | ✖ |
| Generate Instance | ✖ | ✖ | ✓✓ |

- **Compare between UML and feature diagram**

A D&C variability meta model was presented by FD and by UML. FD meta model has proved powerful capability in representation and instantiation. Whereas UML use has proved its inefficiency for textual meta modelling: (1) there is no textual instantiation by specializing a textual model to another, (2) no "or" relation in UML class diagram notation. (3) In class diagram all attributes and operations in primary class will be contained in children classes, this inheritance is not needed in meta modelling. And (4) The attributes and operations do not change from class to another in class diagram. So each child class will contain new text as attribute, generated from the ancestor's classes.

## 5.5 Conclusion: perspectives and future works

This thesis has achieved many contributions. The first consists of a methodology guiding algorithms design by D&C. The second deals with meta modeling D&C variability and formalizing the variability parameters with UML and FD. The third deals with a process guiding the instance algorithm generation. Finally, UML was evaluated to be inappropriate for this kind of meta modeling. This research leaded to the following open problems:

1. The contribution of this thesis is limited to problems having fixed dimensions. A huge number of problem classes are dimension free. Their meta modelling will be very valuable.
2. The combination of Divide & Conquer meta models for dimensioned problems and non-dimensioned ones will be appreciated.
3. The evaluation of the complexity of the proposed methodology (and generally of open layered Meta modelling techniques) relative to limited layers one is a key point in this domain.

4. The UML generalization study to support text-based modelling may lead to its innovation.

# References

Byrne B.M., Yasser Shahzad Qureshi, (2013). THE USE OF UML CLASS DIAGRAMS TO TEACH DATABASE MODELLING AND DATABASE DESIGN. Higher Education Academy.

Chow C., Tsong Yueh Chen, T.H. Tse, (2013). The ART of Divide and Conquer: An Innovative Approach to Improving the Efficiency of Adaptive RandomTesting. Proceedings of the 13th International Conference on Quality Software (QSIC).

Clark T., Andy Evans, Paul Sammut, JamesWillans, (2008). APPLIED METAMODELLING A FOUNDATION FOR LANGUAGE DRIVEN DEVELOPMENT. SECOND EDITION. Ceteva 2008.

Ehrig K., Jochen Malte Küster, Gabriele Taentzer, (2009). Generating instance models from meta models, Software & Systems Modeling, Volume 8, Issue 4, pp 479-500.

Francés J.G., Julio García-Martín, Jose M. Burgos-Ortiz, Miguel Sutil-Martín,(1998). An Approach to Algorithm Design by Patterns.

Giacomo G.D., Maurizio Lenzerini, Riccardo Rosati, (2011). Higher-Order Description Logics for Domain Metamodeling. AAAI.

Gitzel R., T. Hildenbrand, (2005). A Taxonomy of Metamodel Hierarchies. University of Mannheim.

Hao Wu, Rosemary Monahan, James F. Power, (2012). Metamodel Instance Generation:A systematic literature review CoRR abs/1211.6322.

Henderson-Sellers B., (2012). On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer.

Hijma P., Rob van Nieuwpoort, Ceriel J. H. Jacobs, Henri E. Bal, (2011). Automatically Inserting Synchronization Statements in Divide-and-Conquer Programs. IPDPS Workshops 1233-1241.

Jin W., Bo Zhu,Xuanya Li, (2012). A Novel Pipelined Multiplier Using Divide and Conquer Algorithm. International Conference on Industrial Technology and Management.

Kang K.C., Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute Carnegie Mellon University Pittsburgh, Pennsylvania 15213.

Koripadu M., K. Venkata Subbaiah, (2014). Problem Solving Management Using Six Sigma Tools & Techniques. INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 3, ISSUE 2.

K¨uhne T., (2006). Matters of (Meta-) Modeling. Journal on Software and Systems Modeling, Volume 5, Number 4, pp. 369-385.

Lopez-Ortiz A. , Alejandro Salinger, Robert Suderman, (2014). Toward a Generic Hybrid CPU-GPU Parallelization of Divide-and-Conquer Algorithms. IPDPS Workshops 601-610.

Mateos C., Alejandro Zunino, Matías Hirsch, (2013). EasyFJP: Providing hybrid parallelism as a concern for divide and conquer java applications. Comput. Sci. Inf. Syst. (COMSIS) 10(3):1129-1163.

Rahmani M., et al, (2010). A New Design Pattern for Sorting Algorithms. Proceedings of NCSOFT.

Rebori MK., (1995). EFFECTIVE PROBLEM-SOLVIN TECHNIQUES FOR GROUPS. UNIVERSITY OF NEVADA RENO.

Sprinkle J., Bernhard Rumpe, Hans Vangheluwe, and Gabor Karsai, (2010). Metamodelling State of the Art and Research Challenges. Springer-Verlag Berlin Heidelberg.

Witherell P., Anantha Narayanan, JaeHyun Lee, (2011). Using Metamodels to Improve Product Models and Facilitate Inferencing. ICSC 506-513.

vander W.M. P., (2012). Aalst: A General Divide and Conquer Approach for Process Mining. Fed CSIS 2013:1-10.

Zhiyi M., Xiao He, Chao Liu, (2013). Assessing the quality of metamodels. Frontiers of Computer Science (FCSC) 7(4):558-570.

Zuniga M. M, S. Kucherenko, N. Shah, (2013). Metamodelling with independent and dependent inputs. Computer Physics Communications (CPHYSICS) 184(6):1570-1580.

# APPENDIX

The appendix explains, a scenario for the methodology introduced in page 25 (Figure 4-1).

## 1. Divide and Conquer meta model

In the beginning, by applying meta model technique that is applied on divide and conquer method the Figure 1 is obtained.

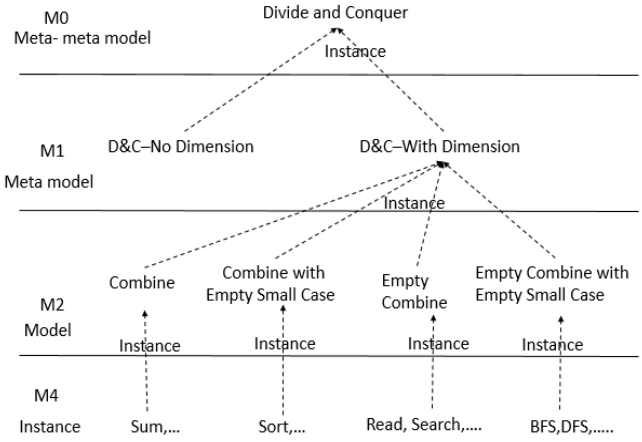



Figure 1. Divide and conquer by meta model technique.

The above divide and conquer meta model (Figure 1), is represented by feature diagram notations as in (Figure 2-A, Figure 2-B).

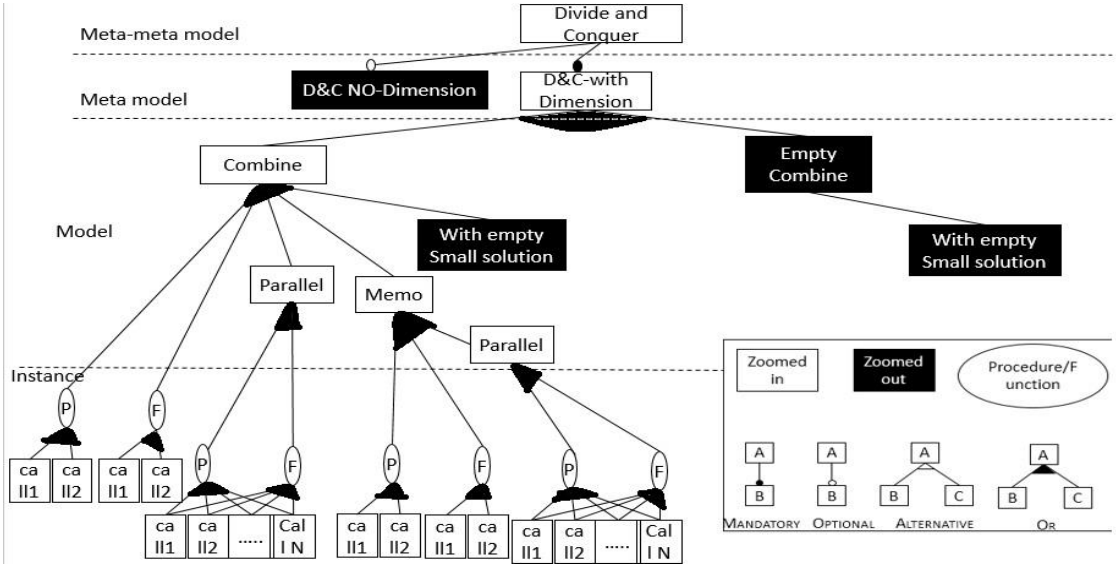



Figure 2-A. Divide and conquer with combine by feature diagram.

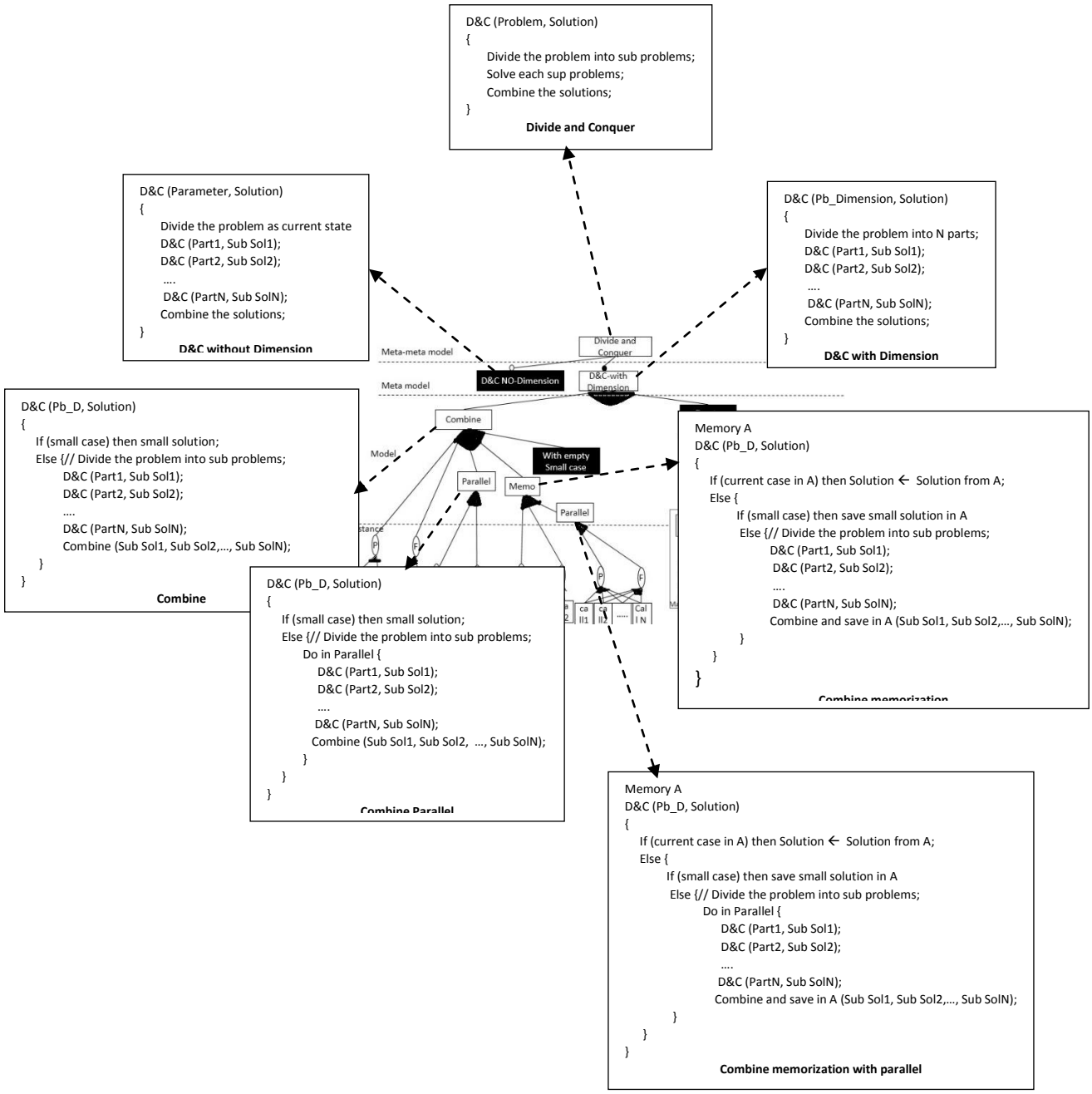



Figure 2-B. Divide and conquer method with combine and Text.

(Figure 2-A) presents a Combine variation FD of D&C method. Combine coming from dimension parameters. It consists of leaves (procedure or function) and many variations features (with empty small case, parallel, memorization, memorization with parallel). Each variation ends by leaves (procedure or function) with the number of call. The (Figure 2-B) shows the Combine variations textual meta model.

## 2. Problem specific algorithm meta model selection request (by user)

The user gives a sentence that describes his problem. This sentence is converted into a selection rule. If this conversion fails, the user may give an example which will be converted into a selection rule. If still there is a failure, the system will produce a default rule (Figure 3).


Figure 3. Problem specific algorithm meta model selection request example

(Figure 3) presents example of problem specific algorithm meta model selection request. The user will give a sentence (sum array) or an example (1, 8, 10, 5, 6 →30), the process will generate the suitable rule (combine/procedure).

## 3. Problem specific algorithm meta model instantiation

Given a selection rule (obtained in Figure 3) and the general meta model (Figure 2-B) the instantiation process will produce the instance candidate for the given problem (Problem model). If specific instance is found, the process will generate it. Otherwise the process will generate a general model.

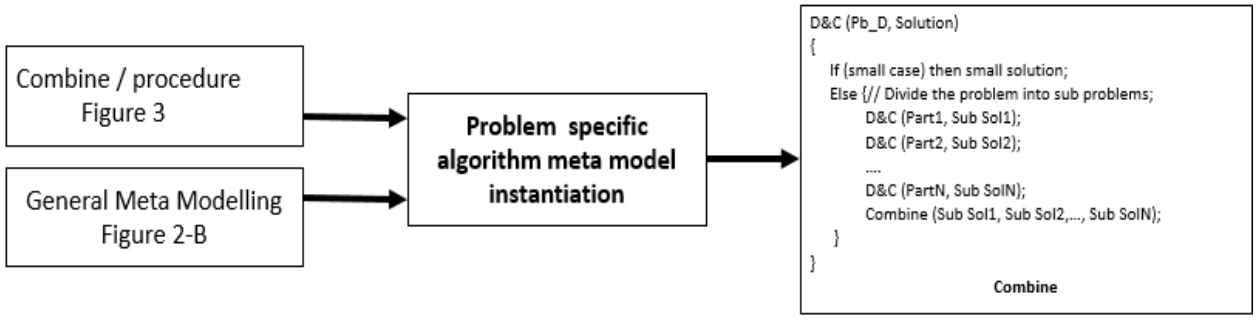



Figure 4. Problem specific algorithm meta model instantiation example.

(Figure 4) presents example of problem specific algorithm meta model instantiation. The process will take selection rule (combine/procedure, obtained in Figure 3) and general meta model (Figure 2-B), the process will generate the suitable model (combine, obtained in Figure 4).

## 4. Problem specific algorithm variability parameters fixing (by user)

This process identifies the variable fields which are the instance parameters for which the user should provide values (Figure 5).

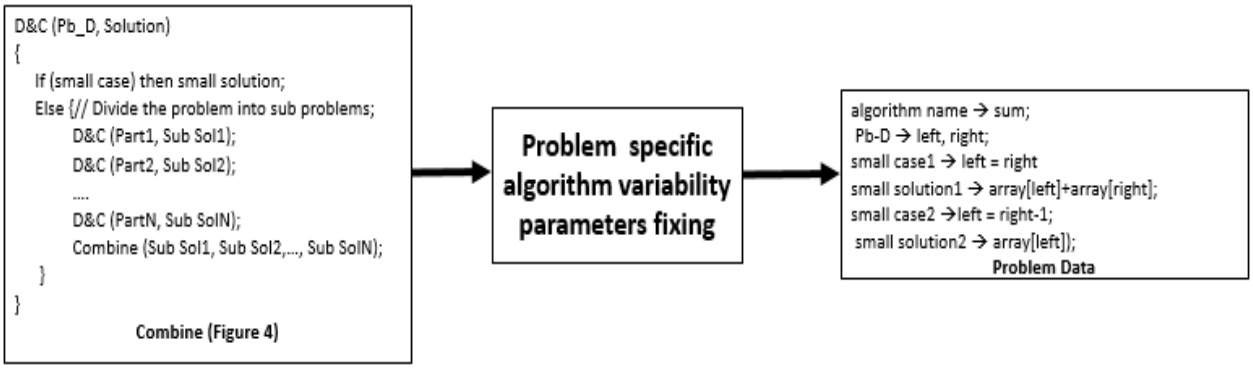



Figure 5. Problem specific algorithm variability parameters fixing example.

(Figure 5) presents example of problem specific algorithm variability parameters fixing. The user will identify the fixed parameters (problem data) for selection model (combine, obtained in Figure 4), the process will take the variable fields from user:

algorithm name→sum;

Pb-D→left, right;

small case1 → left = right

small solution1 → array[left]+array[right];

small case2 →left = right-1;

small solution2 → array[left]);

After the user provides the value, the problem specific algorithm variability parameters fixing process will send the value (in Figure 5) to problem specific algorithm process.

## 5. Problem specific algorithm

This process fills the variable fields of an instance by the user fixed parameters textual values (Figure 6).

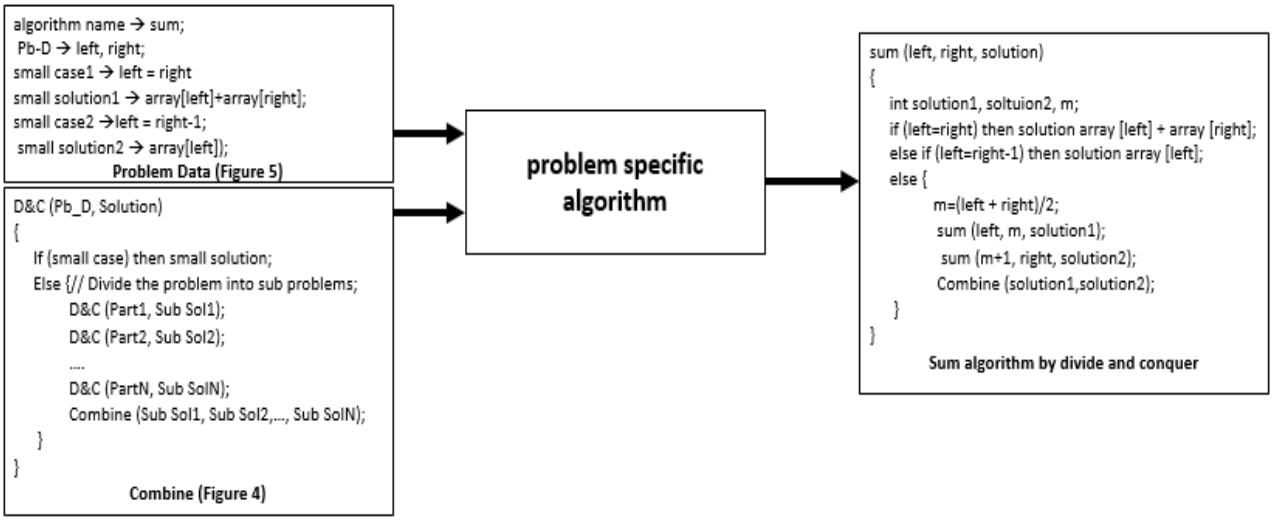



Figure 6. Problem specific algorithm Example

(Figure 6) presents example of problem specific algorithm. The process will fill the fixed parameters (obtained in Figure 5):

algorithm name→sum;

Pb-D→left, right;

small case1 → left = right

small solution1 → array[left]+array[right];

small case2 →left = right-1;

small solution2 → array[left]);

After filling the parameters of an instance (combine, obtained in Figure 4) then will produce sum algorithm by dividing and conquer (obtained in Figure 6).

# ملخص

قسم تسد هي احدى اهم المواضيع في قسم حل المشاكل. تستخدم في عدة مجالات: رياضيات، علوم الحاسوب (كافة مجالاته)، الفيزياء، الهندسة، الخ. بالنتيجة انه موضوع حيوي لأبحاث متنوعة. بعض الابحاث جاءت كتطوير (المعالجة التفريعة، المعالجة التوزيعية، الخ). والبعض الاخر استخدمها في حل مشكلات متعددة في مجالات مختلفة. لكن لا يوجد هناك حتى الان (1) نموذج عام لتمثيل التغيرات في طريقة قسم تسد (2) لايوجد منهجيه تدم هذه التغيرات (3) لايوجد طريقه لاستخلاص خوارزميه خاصه بمشكله معينه (4) عدم دراسة كفائة لغات النمذجه كالتي سوف نقترها كلغة مخطط الميزات. بعض الاعمال حاولت ايجاد نموذج عام لكنها اقتصرت على نطاق محدود جدا. لإيجاد نمط شامل يغطي مجالات مختلفة نحن نحتاج الى مفهوم النمذجه العامة الفوقية وهي جاءت في ابحاث متعددة وواسعه. وهي تملك لغات داعمه مثل لغة النمذجه الموحدة ومخطط الميزات. في اطروحتنا سوف نقترح منهجيه تعتمد على نمثيل بعض التغيرات في طريقة قسم تسد تغطي مشاكل مختلفة في عدة مجالات باستخدام النمذجه الفوقية واللغات الداعمة لها وكيف توليد حل مشكله محدده من النموذج العام. في النتيجه حصلنا على ان مخطط الميزات اكثر استقرارا من لغة النمذجه الموحده لهذا النوع من التغيرات و يمكن تحسين لغة المنذجه الموحده لدعم تغيرات الحاله في طرسقة قسم تسد.

# تمثيل التغيرات في طريقة قسم تسد


بواسطة
احمد فؤاد الجاسم السلطان

بإشـــراف
أ.د. سـعيد الغول


قدمت هذه الرسالة استكمالاً لـمـتطلبات الحصول على درجة
الـــمـاجـستير في علـم الـحـاسوب


عـمـادة البحث العلمي والدراسات العليا
جامعة فيلادلفيا

2014