An Approach to Estimate Design Attributes of Interacting Patterns

D.Janaki Ram, P.Jithendra Kumar Reddy and M.S.Rajasree

***Abstract*— Use of design patterns in designing reusable object-oriented software has increased in recent times. The catalog of object-oriented design patterns [1] contains the details of about twenty three design patterns. A way of representing these design patterns called pattern graph helps in measuring pattern oriented design in terms of some key design attributes like Size, Static Adaptability(SA), Dynamic Adaptability(DA), Extendability(EX). These design attributes measured using pattern graph need to consider the interactions possible between two design patterns, which when neglected may lead to their wrong estimations. This paper gives an overview of the existing pattern measurement model and then explains some of the interactions possible between two design patterns and their impact on pattern oriented designs. A technique for measuring the above mentioned design attributes using pattern graph considering pattern interactions is also being proposed.**

***Index Terms*— Design Patterns, Pattern Oriented Design Attributes(PODAs), Pattern Graphs, Pattern Interactions.**

I. Introduction

MEASURING the software at each phase of development in terms of some attributes is a good practice. Measuring at design phase helps in selecting the best design from a set of design alternatives and in testing whether the design is meeting the quality requirements. Several measurement theories are proposed in [2]. Pertaining to object-oriented design, [3], [4], [5] have proposed several measurement models.

Design patterns which are treated as reusable elements in object-oriented design [1] lack a proper measurement model. The reason is that the design patterns in their present form are not well suited for design measurements . Pattern graph introduced in [6] is a way of representing design pattern which can give a set of direct measures that helps in measuring the design attributes like *Size, Static Adaptability(SA), Dynamic Adaptability(DA) and Extendability(EX)*. We refer these attributes as Pattern Oriented Design Attributes (PODAs).

The present measurement model based on pattern graphs doesn't consider pattern interactions. But, design patterns often interact with other patterns. An approach to see the relationships between design patterns has been made in [7], [8]. According to [7], the categories of relationships possible between two design patterns are *x uses y, x is similar to y, x can be combined with y*. In such cases, calculating the measures for a pattern without considering these relationships may result in errors. These relationships can be seen as interactions between the design patterns as they are influencing the measures. A designer is always interested to have nearly accurate design measures as this helps in taking design decisions, when several design alternatives exist. Hence, pattern interactions influencing the design should be given more attention. Slight modifications to the existing model can take care of the pattern interactions.

In this paper, section II gives a brief overview of measuring pattern oriented design attributes by means of pattern graph with an example as explained in [6]. In section III pattern interactions and their impact is explained. Section IV proposes a measurement model considering pattern interactions with a case study. Section V gives a list of related works. Section VI concludes the paper with a short discussion on the future work.

D.Janaki Ram is an Associate Professor in Department of Computer Science and Engineering, Heads Distributed and Object Systems Lab, P.Jithendra Kumar Reddy is a M.S research scholar under D.Janaki Ram, M.S.Rajasree is a PhD research scholar under D.Janaki Ram, Email:{djram,jithendra,rajasree}@cs.iitm.ernet.in

II. Pattern graph and PODAs

This section gives a brief overview of pattern graph notations, direct measures of a pattern graph, PODAs and their measurements by means of pattern graphs with an example.

A. Overview of pattern graph notations

Figure 1 gives the notations used in pattern graphs. Client class is represented using empty rounded rectangle. Other classes of the pattern are represented using rounded rectangle with three horizontal partitions corresponding to number of template methods, number of hook methods and number of rigid methods respectively. *Hook methods* are those which are declared in a class and defined in the sub-class. *Template methods* are those which call at least one hook method. *Rigid methods* are defined and declared in the same class. They don't call hook or template methods. One-one association between caller and called class is represented by dashed line arrow and one-many association between them is represented using solid line arrow. The dashed line arrow is called as *simple call* and the solid line arrow is called as *composite call*. *Gate* is used to capture inheritance and polymorphism in a less formal way. Gate is associated with a send set and a receive set. Receive set contains the set of classes from whose instances it receives messages and send set contains set of classes, to whose instances it sends messages.

Example pattern graphs for the prototype pattern and composite pattern are given in figure 2. Send set and receive set for the gate in the composite pattern are also shown in figure 2.

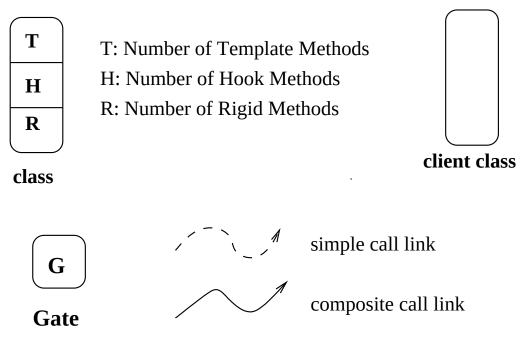
T
H
R
class
T: Number of Template Methods
H: Number of Hook Methods
R: Number of Rigid Methods
client class
G
Gate
simple call link
composite call link

Fig. 1. Pattern Graph Notations

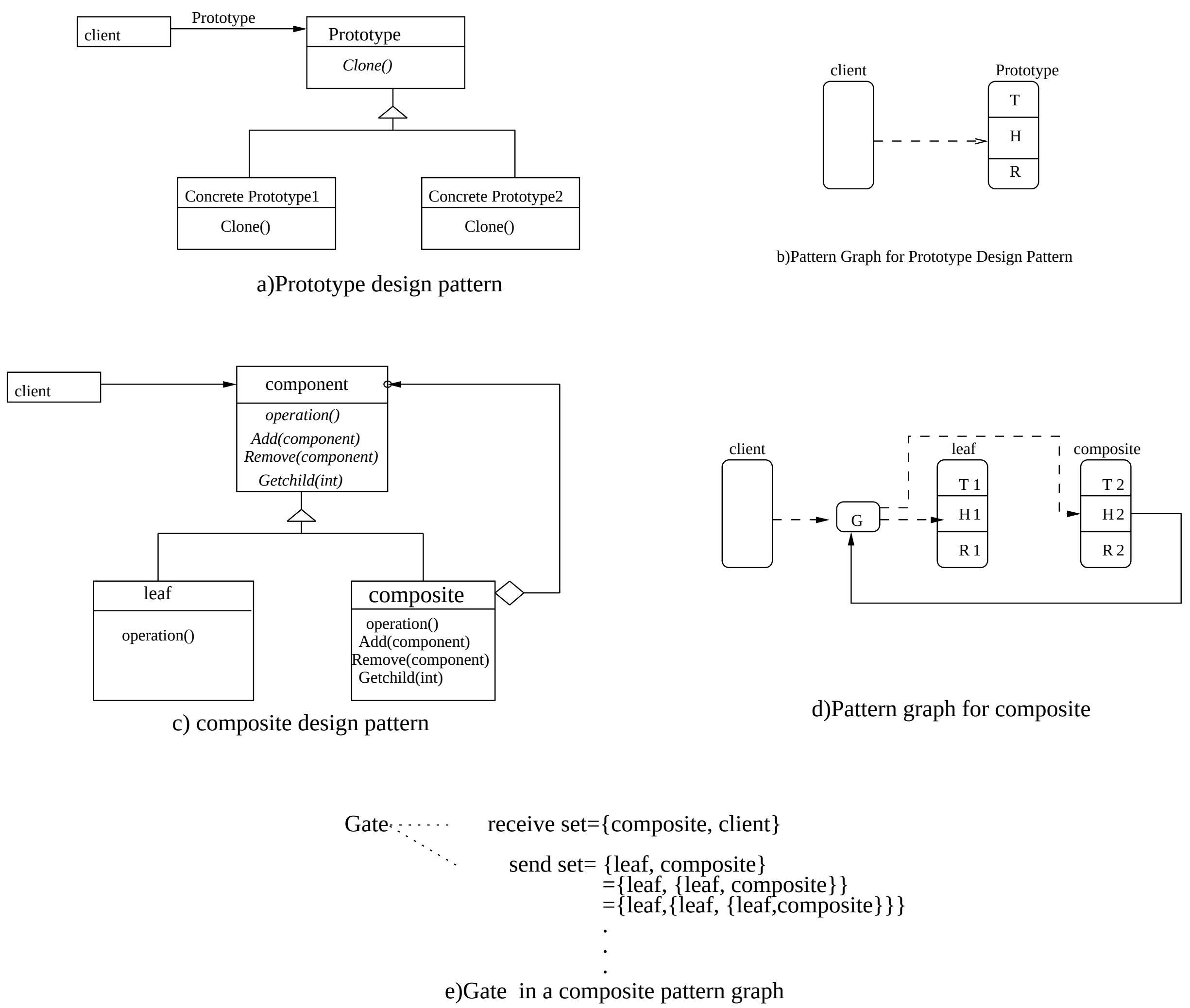

Fig. 2. Example Pattern Graphs

B. Direct measures of a pattern graph

A set of direct measures which can be obtained from a pattern graph are given as follows.

- NC : total number of classes in the pattern graph excluding client class
- NT : total number of template methods in the pattern $NT = \Sigma_{i=1}^{NC} T_i$ where T_i is number of template methods in class i.
- NH : total number of hook methods in the pattern $NH = \Sigma_{i=1}^{NC} H_i$ where H_i is number of hook methods in class i.
- NR : total number of rigid methods in the pattern $NR = \Sigma_{i=1}^{NC} R_i$ where R_i is number of rigid methods in class i.
- SCH : total number of simple calls on hook partitions in the pattern
- CCH : total number of composite calls on hook partitions in the pattern
- CWR : total number of classes which has rigid methods only in the pattern
- CSC : total number of cycles through gate having simple calls only in the pattern
- CCC : total number of cycles through gate having composite calls in the pattern
- CCOC: total number of composite calls outside cycles in the pattern

C. Pattern Oriented Design Attributes (PODAs)

For an OO design, we would like to measure the cost, re-usability and maintainability. Size, Static Adaptability(SA), Dynamic Adaptability(DA) and Extendability(EX) help in measuring them.

Size:
Cost is generally a function of size of the system. For cost calculations, lines-of-code can be treated as unit of measurement of size [9]. As design patterns are identified early in the life cycle, calculating their size helps us to estimate the system size. So, size was measured as

$$Size = w_{SZ,NT} \cdot NT + w_{SZ,NH} \cdot NH + w_{SZ,NR} \cdot NR + w_{SZ,NC} \cdot NC$$

where $w_{SZ,NT}, w_{SZ,NH}, w_{SZ,NR}, w_{SZ,NC}$ are the weights whose values are equal to the average number of lines-of-code for a template method, hook method, rigid method and other aspects of class in the pattern, respectively which are obtained by studying the existing code of similar systems. Since, these values are calculated using an existing code, they relate to a particular programming language and particular domain under consideration. And as such, these weights are likely to converge.

Static Adaptability(SA):
SA is a measure of ease with which a pattern can be adapted to a particular context at the time of coding. Since SA is the adaptation occurring at implementation phase, it is equivalent to reuse by inheritance.

In pattern graph notation, SA is the ease with which one can define a hook method in the sub-class without worrying about defining other hook methods in the pattern [6]. In general, in a hook class with H hook methods, each of the H methods forces us to provide definitions for all H methods during sub-classing. The total number of definitions we are forced to provide while sub-classing, when we consider each of the hook methods one by one is H $\cdot$ H. For a pattern with NC hook classes, the average number of hooks that have to be defined when we try to define one hook is given below
Avg. hooks defined/hook $=(\frac{\Sigma_{i=1}^{NC} H_i^2}{\Sigma_{i=1}^{NC} H_i})$
The reciprocal of this expression is a measure of the contributions of each hook method definition towards itself alone which is termed as SA

$$SA = (\frac{\Sigma_{i=1}^{NC} w_{SA,i} \cdot H_i}{\Sigma_{i=1}^{NC} w_{SA,i} \cdot H_i^2})$$

where H_i represents number of hook methods in class i, and $w_{SA,i}$ is the weight corresponding to class i. One way of calculating this weights is to consider the number of sub-classes of a hook class as a measure of importance of the hook class. Further details of above formula can be obtained from [6].

Dynamic Adaptability(DA):
DA reflects the ease with which the behavior of a pattern can be modified or adapted at runtime [10]. DA is equivalent to reuse by composition as it adapts during runtime. This can be done by replacing objects belonging to the pattern with equivalent objects. In some sense, DA is the measure of extent to which the pattern structure facilitates object composition. In a pattern graph, calls which have a hook partition at their target end represent a set of polymorphic messages. So, the object receiving the messages can be replaced with any other object belonging to any of the set of classes represented by the target class. Also, objects corresponding to classes with rigid methods alone can be replaced with similar objects. So, the measures SCH, CCH, CWR contribute to DA .

$$DA = w_{DA,SCH} \cdot SCH + w_{DA,CCH} \cdot CCH + w_{DA,CWR} \cdot CWR$$

$w_{DA,SCH}, w_{DA,CCH}$ equals to the number of times an object corresponding to the class has been replaced at the target end of simple and composite calls respectively in a given time. $w_{DA,CWR}$ is number of times, objects corresponding to the class having rigid methods have been replaced. These values can be obtained from the existing code after examining over a period of time.

Extendability(EX): EX is the measure of ease with which new objects can be added to the system during the maintenance phase. In a pattern graph, existence of cycles through the gate having only simple calls indicate the possibility of adding new objects to the existing ones of the pattern in a chain like fashion. Similarly, existence of cycles through the gate having only composite calls indicate the possibility of adding new objects in a tree like fashion. Presence of composite calls which are not part of any cycle implies the possibility of adding new objects to form a tree-like structure with one level. So, CSC, CCC, CCOC contribute to EX as explained in [6].

$$EX = w_{EX,CSC} \cdot CSC + w_{EX,CCC} \cdot CCC + w_{EX,CCOC} \cdot CCOC$$

Values of the above weights are calculated from the number of objects which are added to the system in a given period of time by examining the existing code. $w_{EX,CSC}$ is calculated by considering the number of objects added, corresponding to the classes which are in the path of a cycle having simple calls. similarly, $w_{EX,CCC}$ is calculated by considering the number of objects added, corresponding to the classes which are in the path of a cycle having at least one composite call. $w_{EX,CCOC}$ is calculated by considering the number of objects added, corresponding to the classes which are at the target end of a composite call.

All the weights used for calculating above indirect measures can be obtained from the design handbook. The design handbook proposed in [6] is organized into two major sections. First section describes the available design patterns in a suitable format. Section two presents the weights for calculating design attributes. Weights corresponding to various application contexts are available in the design handbook. So, the designer can use the weights from the design handbook which are applicable to the present application context while calculating pattern oriented design attributes.

Calculating the above Pattern Oriented Design Attributes based on design handbook helps a designer to pick a best design from a set of alternatives. A small example of designing a printer system is explained below to explain the importance of calculating PODAs.

• A system for printing files has to be designed. When a user wants to print a file, one from a collection of printers should satisfy his need.

The above problem can be solved in many ways. Two of the design solutions are mentioned below:

1.Design 1: A single print server exists maintaining the details of all printers. User sends the print request to this server. Then this server takes the copy of the requested file and assigns one of the available printers to perform the request.

Mapping patterns to the above design as follows:

a. There exists only one print server: Singleton pattern can be used.

b. File provides a copy of itself to the print server for printing. Let F denote file and PS denote print server :

$PS \rightarrow F$:Prototype pattern is used.

c. Print server forwards request to the available printers. Let PA denotes printer aggregate, which is the collection of printers and PI denotes printer iterator which is used for traversing this collection of printers.

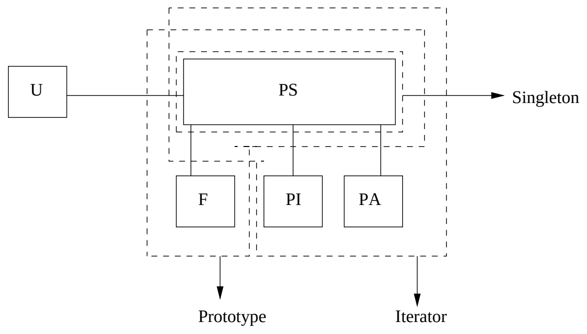

Fig. 3. Design1 for Printer System

$PS \rightarrow^{PI fortraversing} PA$: Iterator pattern is appropriate.

In the first design the set of patterns solving the problem as shown in figure 3 are *Singleton, Prototype and Iterator*.

2.Design 2: Here there exists a chain of printers rather than a single printer. User sends the print request to the first printer in the chain. This request is either accepted by this printer or it is passed on to the next printer. The request traverses along the chain until it is fulfilled. File provides the copy of itself to the printer servicing the request. The set of design patterns that can be mapped to this design are:

a. User sends the request to the printer and this is passed along the chain until it is fulfilled. Let P denotes the collection of printers and U denotes the user.

$U \rightarrow P$: Chain of Responsibility can be used in this context.

b. File Provides a copy of itself to the servicing printer.

$P_{servicingprinter} \rightarrow F$: Prototype pattern is relevant.

In the second design, the set of patterns solving the problem are *Chain of Responsibility and Prototype* as shown in figure 4.

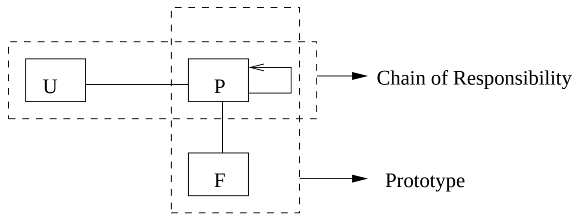

Fig. 4. Design2 for Printer System

Now the problem is to select one of the two alternatives. Calculating the PODAs for each of the designs helps in selecting the best design. Each design level measure is calculated as the sum of the corresponding measures for all the patterns existing in the design.

Overall design measure $X = (\Sigma_{patterni \in design} X of i) + \Delta$ where X can be size, SA, DA,EX and Δ is adjustment value which depends on number of classes in the design which are outside all patterns, as well as the classes which are shared among a set of patterns.

The measures for both the designs are shown in figure 5. The measures are calculated from the C++ sample code for the interface portions of various classes of the patterns which are available in [1]. This helps the designer to pick a design

Designs	Size	SA	DA	EX
I	88	0.43	5	0
II	41	1	2	4

Fig. 5. Experimental Results for the Two Designs

based on his requirements. If he wants his design to be more extendable he can go for design 2, or if he wants his design to be more dynamic adaptable he can go for design 1.

The above mentioned pattern attribute measures are accurate as long as the patterns don't interact internally with other patterns. But, it is often the case a pattern interacts with another pattern. So, the measures mentioned above need to take care of interactions to provide accurate results to the designer. His estimated measures should not far deviate from the actual measures. Issues regarding this are explained in the subsequent sections.

III. Pattern Interactions

There may be several types of interactions between two design patterns. Some of the common interactions possible between two design patterns and their impact are explained in this section.

A. Interaction Types

Frequently occurring pattern interactions can be classified into following types.
Type 1: One pattern *may use* another pattern in it to solve a subproblem.
Type 2: One pattern *may combine* with another pattern for application task completeness.
Type 3: Two patterns solve nearly the same problem in equally valid ways.
Type 4: Distinct patterns share similar structure thus implying a higher level connection.

B. Impact of interactions

The above interactions between design patterns have considerable impact on pattern oriented applications.
1. If the first two types of interactions are not captured in the design, it results in inaccurate measurements of PODAs. Each pattern has a set of design attributes. These attributes are also dependent on the interacting patterns to an extent, which are contained in the main pattern. So, impact of the interactions is likely to affect the pattern attribute measures.
2. Type 3 interaction need to be considered at the elementary pattern selection. Always there exists a trade off between competing patterns.
3. Type 4 interaction has considerable impact while identifying design patterns from existing code. We can retrieve structures from the existing code, but mapping these structures to the patterns becomes difficult due to this interaction.

Pattern attribute measures explained above have been slightly modified to take care of above first two types of interactions. This is explained in the following section.

IV. PODA Measures Considering Pattern Interactions

To calculate Pattern Oriented Design Attributes considering interactions, the set of direct measures required in addition to those specified earlier are

- NPH - number of pattern hooks
  number of hooks in a pattern giving scope for another pattern.
- NNPH - number of non pattern hooks
  number of hooks in a pattern that don't give scope for any other patterns
- SCNPH: simple calls on non pattern hooks
  number of simple calls on hooks not resulting in any other pattern
- CCNPH: composite calls on non pattern hooks
  number of composite calls on hooks not resulting in any other pattern
- NHCP: number of hook chains forming patterns
  it is the number of hook chains each resulting in a pattern. Hook chain is a chain connecting the hook partitions of different classes.
- SCGNP:simple cycles through gate not resulting in any pattern
  it is the number of simple cycles through the gate not resulting in any other pattern.
- CCGNP:composite cycles through the gate not resulting in any pattern
  it is the number of composite cycles through the gate not resulting in any other pattern.

A. x uses y interaction:

A pattern uses other pattern to solve one of its sub problem. A hook method in a pattern provides a scope for using another pattern. Pattern may have to use a different pattern while providing definitions to the hook method in the sub-class depending on the application context. Consider , prototype pattern in the file management context. Master file is a file which can give only one copy of it. So, prototype pattern should use singleton pattern while providing definition to the clone hook method in the Master file sub-class as shown in figure 6. In other words, prototype pattern is using singleton pattern while solving the problem of sub-classing master file.

So, in order to take care of x uses y interaction while measuring PODAs, a slight modification needs to be done in the formulas as explained below. These modifications account for the measures resulted from the new patterns introduced in the design while definitions are provided for hooks.

Size: For calculating size, we need to consider the size of sub pattern resulted by each pattern hook.

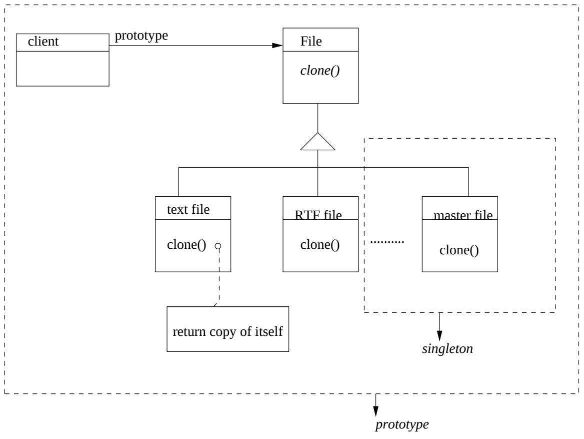

Fig. 6. Prototype Using Singleton

$$size = w_{SZ,NT} \cdot NT + w_{SZ,NNPH} \cdot NNPH + w_{SZ,NR} \cdot NR + w_{SZ,NC} \cdot NC + \Sigma_{i=1}^{NPH} Size of P_i$$

In the example shown in figure 6, there exists one pattern hook , clone() resulting in singleton pattern. Clone() methods in text file and RTF file in figure 6 are considered as non pattern hooks.

Static Adaptability(SA): Effective SA of a pattern needs to include the SA of the pattern used.

$$SA = \left(\frac{\Sigma_{i=1}^{S} w_{SA,i} \cdot NNPH_i}{\Sigma_{i=1}^{S} w_{SA,i} \cdot NNPH_i^2}\right) + \Sigma_{i=1}^{NPH} SA of P_i$$

Dynamic Adaptability(DA): As explained earlier, it is a function of SCH, CCH, CWR. So, simple and composite calls on non pattern hooks and DA of patterns resulted by pattern-hooks contribute to effective DA.

$$DA = w_{DA,SCNPH} \cdot SCNPH + w_{DA,CCNPH} \cdot CCNPH + w_{DA,CWR} \cdot CWR + \Sigma_{i=1}^{NPH} DA of P_i$$

Extendability(EX): Extendability mainly depends on the cycles passing through the gate. It is often the case, that a sub pattern used contains no cycles through gate. But there are cases , where a hook may result in a pattern which contains cycles. The case study explained in later section has this result. So, the effective EX considering the uses interaction is

$$EX = w_{EX,CSC} \cdot CSC + w_{EX,CCC} \cdot CCC + w_{EX,CCOC} \cdot CCOC + \Sigma_{i=1}^{NPH} EX of P_i$$

B. *x combines with y interaction:*

A pattern combines with other pattern, both solving different problems, but are combined to solve a particular application task. Existence of chain of hooks in a pattern gives scope for combining with other pattern. Consider the example shown in figure 7, roles instantiating corresponding privileges list, forms a Factory Method pattern, while iterator pattern is used for traversing each of the privileges list.

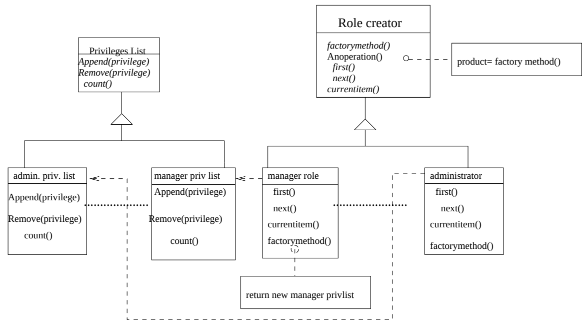

Fig. 7. Factory Method Combined with Iterator

Factory Method is solving the problem of instantiating an object at sub-class level, whereas iterator pattern solves the problem of traversing through the instantiated privileges list. Both solving different problems, but iterator pattern is combined with in the Factory Method for completing the application task. Figure 8 shows the pattern graph of Factory Method. The chain of hook methods in Factory Method pattern graph as shown in the figure 8 gives scope for combining with iterator pattern.

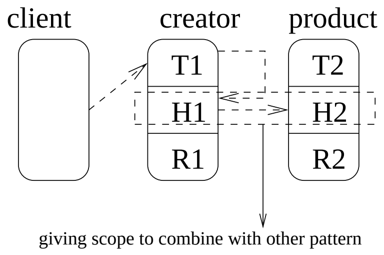

Fig. 8. Pattern Graph for Factory Method

Size: Size of the pattern resulted by a hook chain should also be considered

$$Size = w_{SZ,NT} \cdot NT + w_{SZ,NNPH} \cdot NNPH + w_{SZ,NR} \cdot NR + w_{SZ,NC} \cdot NC + \Sigma_{i=1}^{NHCP} Size of P_i$$

Static Adaptability: As explained for size, for calculating the effective SA of the pattern, we need to include the SA of patterns resulted by the hook chains in addition.

$$SA = \left(\frac{\Sigma_{i=1}^{S} w_{SA,i} \cdot NNPH_i}{\Sigma_{i=1}^{S} w_{SA,i} \cdot NNPH_i^2}\right) + \Sigma_{i=1}^{NHCP} SA of P_i$$

Dynamic Adaptability: The explanation is same as that of *x uses y* interaction.

$$DA = w_{DA,SCNPH} \cdot SCNPH + w_{DA,CCNPH} \cdot CCNPH + \Sigma_{i=1}^{NHCP} DA of P_i$$

Extendability: In addition to the EX of the main pattern, EX of the patterns resulted by some of the hook chains contribute to effective Extendability .

$$EX = w_{EX,SCGNP} \cdot SCGNP + w_{EX,CCGNP} \cdot CCGNP + \Sigma_{i=1}^{NHCP} EX of P_i$$

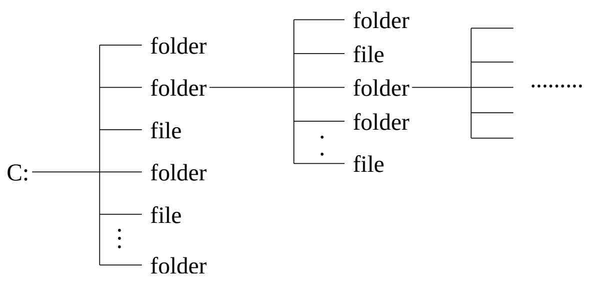

Fig. 9. Searching Folder Hierarchy

Weights for the above mentioned formula are calculated in a similar way which was proposed in [6] and can be made available in the design handbook corresponding for each pattern.

C. Case Study:

This section gives a small case study of searching an item in a folder hierarchy as shown in figure 9. The problem is to search a given file or folder in a particular partition, C partition in this case as shown in figure 9. C partition consists of a list of files and folders and each folder may contain a folder or file. Pattern oriented design for solving this problem is shown in figure 10. The explanation of the design is as follows
1. In the first layer , a list of folders and files exist. Iterator pattern is used to traverse through this list for searching the required item.
2. Each folder item can be designed using composite pattern. The recursive abstraction of folder(a folder containing a file or folder, and the second level folder may again contain a file or folder, which happens recursively)is captured by composite pattern. Iterator pattern is combined with composite pattern for traversing through the hierarchy and search the required item as shown in figure 10.

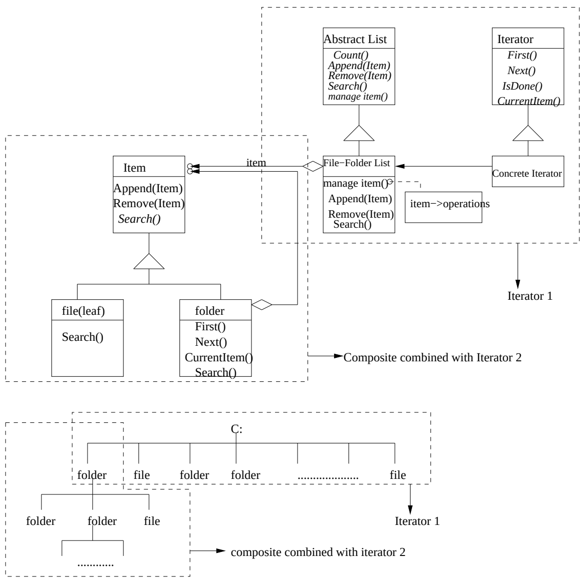

Fig. 10. Pattern Oriented Design for Searching Folder Hierarchy

Now the entire design can be put as

Iterator 1 *uses*(composite *combined with* Iterator 2).

Now the calculations of Pattern Oriented Design Attributes for the above case is as follows:

$$size = w_{SZ,NT_{iterator1}} \cdot NT_{iterator1} + w_{SZ,NNPH_{iterator1}} \cdot NNPH_{iterator1} + w_{SZ,NR_{iterator1}} \cdot NR_{iterator1} + w_{SZ,NC_{iterator1}} \cdot NC_{iterator1} + \Sigma_{i=1}^{NPH} SizeofP_i$$

$$=> size = w_{SZ,NT_{iterator1}} \cdot NT_{iterator1} + w_{SZ,NNPH_{iterator1}} \cdot NNPH_{iterator1} + w_{SZ,NR_{iterator1}} \cdot NR_{iterator1} + w_{SZ,NC_{iterator1}} \cdot NC_{iterator1} + (w_{SZ,NT_{composite}} \cdot NT_{composite} + w_{SZ,NNPH_{composite}} \cdot NNPH_{composite} + w_{SZ,NR_{composite}} \cdot NR_{composite} + w_{SZ,NC_{composite}} \cdot NC_{composite} + \Sigma_{i=1}^{NHCP} SizeofP_i)$$

$$=> size = w_{SZ,NT_{iterator1}} \cdot NT_{iterator1} + w_{SZ,NNPH_{iterator1}} \cdot NNPH_{iterator1} + w_{SZ,NR_{iterator1}} \cdot NR_{iterator1} + w_{SZ,NC_{iterator1}} \cdot NC_{iterator1} + (w_{SZ,NT_{composite}} \cdot NT_{composite} + w_{SZ,NNPH_{composite}} \cdot NNPH_{composite} + w_{SZ,NR_{composite}} \cdot NR_{composite} + w_{SZ,NC_{composite}} \cdot NC_{composite} + (w_{SZ,NT_{iterator2}} \cdot NT_{iterator2} + w_{SZ,NH_{iterator2}} \cdot NH_{iterator2} + w_{SZ,NR_{iterator2}} \cdot NR_{iterator2} + w_{SZ,NC_{iterator2}} \cdot NC_{iterator2}))$$

In this case number of pattern hooks(NPH) of ”iterator 1” is one. manageitem() is the hook of ”iterator 1” resulting in composite pattern as shown in figure 10. Similarly, number of hook chains(NHCP) for ”composite” is one which results in ”iterator 2”. The other pattern attributes are calculated in the same way as shown above.

In the existing model proposed in [6], if hooks are giving scope to other patterns, then there is no way of considering the design attributes of these patterns. But in the present model, there is a clear distinction between hooks not giving scope to any patterns and hooks giving scope to patterns. The design attributes of the patterns resulted through hooks are also considered. For above example, designer can get the weights corresponding to ”iterator 1”, ”composite” and ”iterator 2” from design handbook whose values are obtained from the past experience of similar cases. So, a designer can almost estimate his design level measures accurately using the proposed measurement model.

V. Related works

A few relationships between the design patterns is given in [7]. [8] also describes and classifies the common relationships between object-oriented design patterns. In the present paper, a measurement model for pattern oriented designs considering such relationships is being proposed.

Meta patterns are described for a set of design patterns that describes how to construct frameworks independent of a specific domain in [11]. Pattern graph explained in the paper has some features of meta patterns, but the perspective in which it is viewed is towards a pattern oriented design measurement model, which is never addressed by meta patterns [11].

In [6], a notion of pattern graph is introduced for measuring the patterns in terms of some key attributes. In the present paper, pattern attributes are measured by means of pattern graphs considering interactions which is not addressed in [6].

VI. Conclusions and future work

Design patterns have been recognized as a means of recording the designer's experience. They serve as reusable elements while constructing object- oriented software. Measuring these patterns provide an insight into the design. So, the measurement should be as accurate as possible. Towards the direction of accurate measurement, pattern interactions are considered in the measurement model.

A mechanism to capture first two types of mentioned interactions is explained. The third type of interaction results in the problem of selecting one of the two alternatives, which has already been solved by the pattern graph. However, the last type of interaction whose impact falls on identifying design patterns from the existing code is not captured. More focus needs to be made on this direction.

The metrics proposed need to be tested with a concrete example. Currently, we are working on developing a methodology for selecting a best pattern quickly. As a part of future work, we are trying to propose a systematic way for developing complex architectures from patterns.

References

[1] Gamma E, Helm R, Johnson R, and Vlissides J, *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[2] Fenton N, "Software Measurement: A Necessary Scientifi c Basis," *IEEE Transactions on Software Engineering*, vol. 20, no. 3, pp. 199–206, 1994.

[3] Whitmire S A, *Object-Oriented Design Measurement*. Wiley, New York, 1997.

[4] Shyam R. Chidamber and chris F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[5] Rachel Harrison and Steve J. Counsell, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 491–496, 1998.

[6] Janaki Ram D, Anantharaman K N, Guruprasad K N, Sreekanth M, Raju S V G K, and Ananda Rao A, "An approach for pattern oriented software development based on a design handbook," *Annals of Software Engineering*, vol. 10, pp. 329–358, 2000.

[7] W. Zimmer, "Relationships between design patterns,in pattern languages of program design," pp. 345–364, Addison-Wesley, 1994.

[8] Noble J, "Classifying relationships between object-oriented design patterns," in *proceedings of Australian Software Engineering Conference (ASWEC)*, 1998.

[9] Boehm B H, "Software Engineering Economics," *IEEE Transactions on Software Engineering*, vol. 10, pp. 135–152, 1984.

[10] Janaki Ram D, Anantharaman K N, and Guruprasad K N, "A Pattern Oriented Technique for Software Design ," *ACM Software Engineering Notes*, vol. 22, no. 4, pp. 70–73, 1997.

[11] Wolfgang Pree, *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, 1995.