ISSN 2278 - 3091

**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol.2 , No.5, Pages : 05-08 (2013)
*Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India*

# Application building concepts in medical image processing using software design patterns


**Niranjan R Chougala,** B.E., M.Tech., [ Ph.D.-VTU ]
Assistant Professor, Information Sciences & Engg. Dept.,
HKBK College of Engineering, S.No. 22/1, Nagawara, Bangalore-45, Karnataka
Email : nrchougala@gmail.com Phone : +91 9916132205

**Dr. Shreedhara K.S.**
Professor & Chairman DOS in CS & E,
University BDT College of Engineering – Davanagere, Karnataka

**Abstract—The next generation software in particular to medical domain should able to adopt the new changes and conditions that may arise when the software's are deployed. Since, medical image processing and medical software's are complex and difficult, thereby making a highly adaptive system, a necessary. This paper mainly focus on the various approaches towards developing a medical software's in particular to medical image processing using Software design patterns, thereby making it possible to design at a higher level of abstraction and standardize various design patterns which may be useful in near future for the medical software developers. It can be design, class or image patterns, reuse of the code remain the prime focus by building software architecture and design using new approach. This approach not only provides a new dimension to the medical image processing approach but also builds standard signature and communication interfaces among the softwares. Thus, the purpose of developing and standardizing efficient, robust, extensible, standard and reusable software remains a challenge.**

***Keywords* — Medical Image Processing, Adaptive systems, Design Patterns.**

## I. INTRODUCTION

In general, we all need efficient, robust, extensible, and reusable software, which is difficult and complex. The use of design patterns helps to reduce the complexity and makes design reuse. Successful developers resolve these challenges by applying appropriate design patterns. Thus, Design patterns represent solutions to problems that arise when developing software within a particular context. Continuous advances in the size and functionality of medical imaging techniques and processes over recent years has resulted in an increasing interest in their use as implementation platforms for image processing applications.

The physical structure of the design allows us to exploit the parallelism inherent in low-level image processing operations. This parallelism exists in two major forms

1. Spatial parallelism, in which the image is divided into multiple sections and processed concurrently, and

2. Temporal parallelism, where the algorithm may be represented as a time sequence of simple concurrent operations.

These implementations have the potential to be parallel using a mixture of these two forms. Pragmatically, the degree of parallelization is subject to the processing mode and hardware constraints imposed by the system [1]. Based on previous work [2], [3] we believe there are three processing modes: stream, offline and hybrid processing. We have also identified the following constraints: timing (limited processing time), bandwidth (limited access to data), and resource (limited system resources) constraints. These constraints are inextricably linked and manifest themselves in different ways depending on the processing mode. Managing constraints makes the mapping of image processing algorithms to hardware more challenging. atterns are about communicating problems and solutions. In other words, patterns enable us to document a known recurring problem and its solution in a particular context and to communicate this knowledge to others. Here, main focus is on the term recurring, since the goal of the pattern is to foster conceptual reuse over time [4]. Some of the common characteristics of patterns are as follows [4].

- Patterns are observed through experiences.
- Patterns are typically written in structural format.
- Patterns prevent reinventing the wheel.
- Patterns exist at different levels of abstraction.
- Patterns undergo continuous improvements.
- Patterns are reusable artifacts.
- Patterns communicate and designs best practices.
- Patterns can be used together to solve a large problem.
- Design patterns represent solutions to problems that arise when developing software within a particular context.
- Patterns capture the static and dynamic structure and collaboration among key participants in software designs.
- Patterns facilitate reuse of successful software architectures and designs.

Categorizing Patterns [4] then represents the expert solutions to recurring problems in a context and thus have

ISSN 2278 - 3091

**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol.2 , No.5, Pages : 05-08 (2013)
*Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India*

been captured at many levels of abstraction and in numerous domains. Numerous categories have been suggested for classifying software patterns, with some of the most common being:

- Design Patterns
- Architectural Patterns
- Analysis Patterns
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

The J2EE Patterns [4] included in the following table classify each pattern with respect to one of the three logical architectural tiers, namely, Presentation, Business and Integration.

TABLE I : J2EE PATTERNS

| **Tier** | **Pattern name** |
|---|---|
| Presentation | Intercepting Filter<br>Front Controller<br>Context Object<br>Application Controller<br>View Helper<br>Composite View<br>Service to Worker<br>Dispatcher View |
| Business | Business Delegate<br>Service Locator<br>Session Façade<br>Application Service<br>Business Object<br>Composite Entity<br>Transfer Object<br>Transfer Object Assembler<br>Value List Handler |
| Integration | Data Access Object<br>Service Activator<br>Domain Store<br>Web Service Broker |

However, one of the challenges when using any set of patterns is …. Understanding how to best use the pattern in combination. According to Christopher Alexander (in his book Patterns Language) "In short, no pattern is an isolated entity. Each pattern can exist in the world, only to the extent that is supported by other patterns: the larger patterns in which it is embedded and the patterns of the same size that surround it and the smaller patterns which are embedded in it". J2EE patterns are no exception to this rule. As we study each of these patterns in detail, we will see the patters and strategies that are embedded within it, in which it is contained, and which it supports. Sometimes these patterns build on other patterns from the J2EE patterns catalog or from others. Benefits of using patterns:

- Leverage a proven solution
- Provide a common vocabulary
- Constrain solution space

There are various relationships between the patterns, and these are generally referred to as being part of Pattern Language. Also, there is another way of defining these relations is in terms of Pattern Framework, that is, collection of patterns in a united scenario. This is a key factor in identifying end-to-end solutions and wiring components together at the pattern level.

## II. VB.NET DESIGN PATTERNS

In Visual Basic [6] developers recognizes more benefits and they brought most of the useful features of the concepts to the users. Also, the arrivals of the .NET frameworks and VB.NET has dramatically changed the analysis, since it is truly object-oriented, and therefore good choice of production language for OO applications whose designs are on design patterns. Visual developers have acquired one of the most powerful object-oriented languages and also discovered that migration from VB6 to VB.NET is more difficult than their migration to the earlier versions of VB, it is armed with a mastery of API calls. However VB.NET can now able to provide a vital solutions to some of the challenges faced by the developer community. Since, this paper restricts it study to the Medical domain image processing, focus and study is restricted to its limit.

## *III.* WHAT, WHERE AND WHERE NOT ABOUT DESIGN PATTERNS

*Design Pattern Descriptions* [7]

- Name and Classification: Essence of pattern
- Intent: What it does, its rationale, its context
- AKA: Other well-known names
- Motivation: Scenario illustrates a design problem
- Applicability: Situations where pattern can be applied
- Structure: Class and interaction diagrams
- Participants: Objects/classes and their responsibilities
- Collaborations: How participants collaborate
- Consequences: Trade-offs and results
- Implementation: Pitfalls, hints, techniques, etc.
- Sample Code
- Known Uses: Examples of pattern in real systems
- Related Patterns: Closely related patterns

Design Patterns are useful where …

- Object-Oriented programming languages [and paradigm] are more amenable to implementing design patterns
- Procedural languages: need to define
  - Inheritance
  - Polymorphism
  - Encapsulation

and they are NOT …

- Designs that can be encoded in classes and reused as is (i.e., linked lists, hash tables)

ISSN 2278 - 3091

**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol.2 , No.5, Pages : 05-08 (2013)
*Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India*

- Complex domain-specific designs (for an entire application or subsystem)
- They are: "Descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context."

They are USED in…

- Solutions to problems that recur with variations
  - No need for reuse if problem only arises in one context
- Solutions that require several steps:
  - Not all problems need all steps
  - Patterns can be overkill if solution is a simple linear set of instructions
- Solutions where the solver is more interested in the existence of the solution than its complete derivation
  - Patterns leave out too much to be useful to someone who really wants to understand
  - They can be a temporary bridge

What Makes Patterns ?

- A Pattern must [7] :
  - Solve a problem and be useful
  - Have a context and can describe where the solution can be used
  - Recur in relevant situations
  - Provide sufficient understanding to tailor the solution
  - Have a name and be referenced consistently

## IV. BENEFITS, DRAWBACKS & EFFECTIVE USE

The research work undertaken in this paper has the following benefits and drawbacks.

BENEFITS:

- Design patterns enable large-scale reuse of software architectures and also help document systems
- Patterns explicitly capture expert knowledge and design tradeoffs and make it more widely available
- Patterns help improve developer communication
- Pattern names form a common vocabulary
- Patterns help ease the transition to OO technology

DRAWBACKS:

- Patterns do not lead to direct code reuse
- Patterns are deceptively simple
- Teams may suffer from pattern overload
- Patterns are validated by experience and discussion rather than by automated testing
- Integrating patterns into a software development process is a human-intensive activity.

However, The suggestion for the effective use includes…

- Do not recast everything as a pattern
  - Instead, develop strategic domain patterns and reuse existing tactical patterns
- Institutionalize rewards for developing patterns
- Directly involve pattern authors with application developers and domain experts
- Clearly document when patterns apply and do not apply
- Manage expectations carefully.

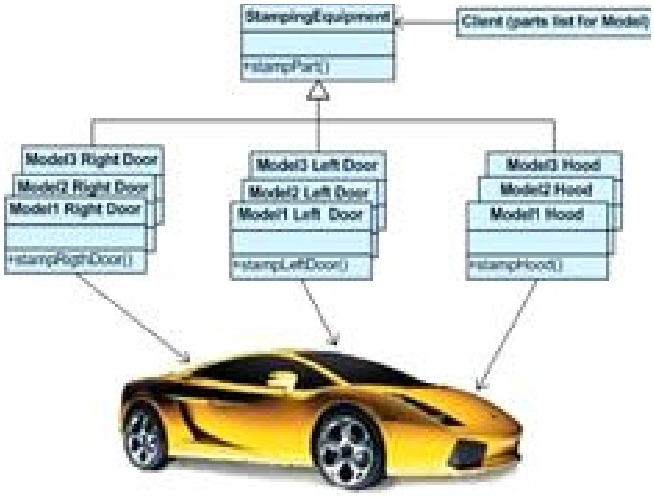



Fig. 1 : Creational design patterns

## V. EXAMPLES

http://sourcemaking.com/design_patterns

*Creational design patterns:*

This design patterns is all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

1. **Abstract Factory** - Creates an instance of several families of classes
2. **Builder** - Separates object construction from its representation
3. **Factory Method** - Creates an instance of several derived classes
4. **Object Pool** - Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
5. **Prototype** - A fully initialized instance to be copied or cloned
6. **Singleton** - A class of which only a single instance can exist

*Structural design patterns:*

This design patterns is all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

ISSN 2278 - 3091

**International Journal of Advanced Trends in Computer Science and Engineering (IJATCSE)**, Vol.2 , No.5, Pages : 05-08 (2013)
*Special Issue of ICCECT 2013 - Held during September 20, 2013, Bangalore, India*

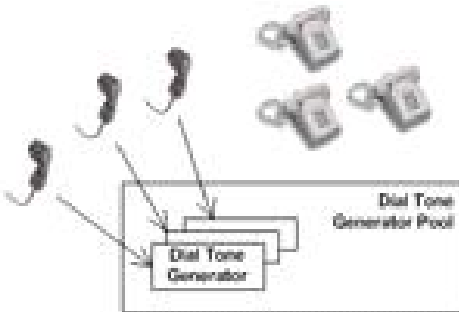
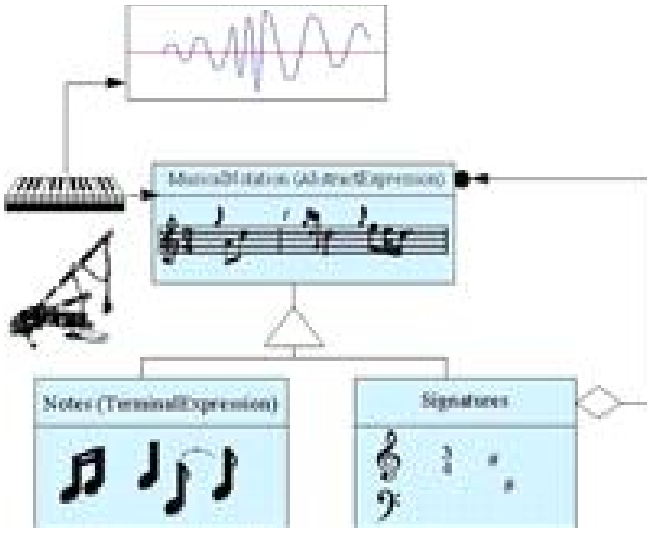



Fig. 2 : Structural design patterns

**Adapter -** Match interfaces of different classes
**Bridge** - Separates an object's interface from its implementation
**Composite -** A tree structure of simple and composite objects
**Decorator -** Add responsibilities to objects dynamically
**Façade -** A single class that represents an entire subsystem
**Flyweight -** A fine-grained instance used for efficient sharing
**Private Class Data -** Restricts accessor/mutator access
**Proxy -** An object representing another object

*Behavioral design patterns:*
This design patterns is all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.



**Chain of responsibility -** A way of passing a request between a chain of objects
**Command -** Encapsulate a command request as an object
**Interpreter** - A way to include language elements in a program
**Iterator -** Sequentially access the elements of a collection
**Mediator** - Defines simplified communication between classes
**Memento -** Capture and restore an object's internal state
**Null Object -** Designed to act as a default value of an object
**Observer -** A way of notifying change to a number of classes
**State -** Alter an object's behavior when its state changes
**Strategy -** Encapsulates an algorithm inside a class
**Template method -** Defer the exact steps of an algorithm to a subclass
**Visitor -** Defines a new operation to a class without change

## VI. CONCLUSION

The rapid changes and technological advancements in the Software Engineering methodologies and medical domain make it necessary to adapt next generation ready applications. This survey and study primarily focuses on the use of medical application developments through Software Design Patterns and to standardize some of the design patterns as applicable to the medical software development area. The survey indicates a larger scope of application of these design patterns in the said domain and primarily highlights the need of convergence with consensus with software and medical domains. This study also refines the fact that, with more emphasis on design patterns the said domains can be more benefited in terms of efficiency, robust, extensible, reliable, reusable and internationalization.

## REFERENCES

[1]. K. T. Gribbon, D. G. Bailey, C. T. Johnston, "Using Design Patterns to Overcome Image Processing Constraints on FPGAs", Institute of Information Sciences and Technology Massey University, Private Bag 11 222, Palmerston North, New Zealand. 2006.

[2]. Gribbon, K. T. and Bailey, D. G., "A Novel Approach to Real-time Bilinear Interpolation," Second IEEE International Workshop on Electronic Design, Test and Applications, Perth, Australia, pp. 126-131, Jan, 2004.

[3]. Gribbon, K. T., Johnston, C. T., and Bailey, D. G., "A Realtime FPGA Implementation of a Lens Distortion Correction Algorithm with Bilinear Interpolation," Proc. Image and Vision Computing New Zealand, Massey University, Palmerston North, New Zealand, pp. 408-413, Nov, 2003.

[4]. Deepak Alur, John Crupi and Dan Malks, "Core J2EE" patterns, Second Edition, 2003.

[5]. Douglas C. Schmidt, "Using Design Patterns to Develop Object-Oriented Communication Software Frameworks and Applications", Washington University, St. Louis.

[6]. Tom Fischer, John S, Pete S, Chaur G Wu "Professional Design Patterns in VB.NET, Building Adaptable Applications", Wrox Press, 2002.

[7]. Gama, Helm, Johnson, Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, Addison Wesley, 1995,B. Cheng – Michigan State University.