M O N A S H   U N I V E R S I T Y

Faculty of Information Technology
School of Computer Science and Software Engineering

CSE5510 Software System Engineering

Second Semester, 2002

Written Assignment

Pattern-Oriented Software Architecture

Kim Guan, CHUA (18218792)

Lecturer: Christine Mingins

Due Date: Monday 23th October 2002

Pattern-Oriented Software Architecture

Kim Guan, CHUA
chuakimguan@hotmail.com

Abstract

The way of building software is changing rapidly. A decade ago, choosing a right algorithm and data structure is imperative, it determines if the entire system works as it expects. However, nowadays, there are many common standard libraries and components to ease developing software and handle the data structure and algorithm for the programmers. Programmers often do not need to write their own algorithm or data structure unless there are special purpose application. Moreover, the compiler is smart to do most of the tasks for the programmer, such as optimizing. Software architect can focus more time on the structure of the entire system. Therefore, software architecture is getting more attendance. This paper presents an overview of software architecture and discusses some of the common architectural patterns.

Introduction

“I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.”

The above phase is from Frederick’s paper "No Silver Bullet: Essence and Accidents of Software Engineering” published in IEEE Computer 1987. It is well known that one of the hardest parts of building software is deciding the structure of the whole system and define the relationship of its components. There are many new researches conducted in this field, such as formal approaches to software architecture [4], pattern-oriented software architecture [5].

This paper first introduces what software architecture and pattern are in software engineering term. Then, It discussed how software architecture and pattern fitted together, which is pattern-oriented software architecture. Finally, it presents some of the common architectural patterns and its usage.

Software Architecture

The software architecture of an application is the structure or a collection of structures, which consists of those components, externally visible properties of those components and the relationship among them [1]. Software architecture is the blueprint of the entire system. It answers what the entire system can and cannot do before the system is constructed. Therefore, it reduces the risks of construction the system. Building software without concern of the architecture is like building a house without a blueprint, which is extremely risky. Having a complete architecture provides you a better or precise overview of the whole system.

Pattern

The original idea of pattern rose from Alexander's patterns of building architecture. According to him [2] "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution." In simple term, pattern is a document, which describes a design problem and a general solution of it in a particular context [8]. It was proved to be one of the best approaches, and not only apply to building houses and apartments; but also software. Pattern is the results of many experience engineers or experts in that particular field. Software community adopts pattern in early 80. On of the famous pattern book, Design pattern gains many audiences.

Architecture pattern is also called architectural style in some architecture literatures [6] [7]. It is common found that one or more architectural patterns have been adopted as strategies for system organization. Moreover, these patterns have usually been used informally and explicitly [6] and most software systems have been built using more than one architectural pattern.

A pattern system provides on one hand, a collection of proven solutions to many recurring design problems. On the other hand, it demonstrates how to combine individual patterns into heterogeneous structures and as such it can be used to smooth the progress of a constructive development of software systems.

Classifying Patterns

According to Coplien, the early communities have separate pattern into 3 layers, i.e., architectural patterns/framework pattern, Design pattern, and idiom [8]. The idiom is the lowest level of pattern. It describes how the code can be implemented in particular programming language. The design pattern describes how to solve problem by providing a collection of class or subsystem and define the relationship among them. Architectural pattern is the highest level of abstraction in software patterns; it defines the structure of the whole system and its relationship. However, there is limitation on this classification since some patterns are hard to categorize due to its broad, e.g. Model-View-Controller [8].

Pattern and software architecture

Pattern-Oriented Software Architecture is an alternative approach to software development. It expresses fundamental structural organization schemas for software system. As discussed in previous section, architectural patterns represent the highest-level patterns of pattern system. Design patterns and idiom solve coding problem whereas architecture patterns address the problem on building software. Architectural patterns such as layered patterns are not new at all. They might be used in telecommunication applications for few decades before someone discover them and document them. According to pattern-oriented software architecture book POSA [5], architectural patterns can be grouped into 4 categories. The following section presents brief description of common architectural patterns used today. They derive heavily from POSA [5].

From Mud to Structure

The Patterns in this category supports a controlled dismantles of a whole task of system into cooperating components, subsystems or layers. There are three patterns fall in this category, i.e. layers pattern, Blackboard pattern and “Pipes and Filters” pattern.

1. Layers Pattern
   The layers pattern decomposes the tasks into different layer and each layer is at a particular level of abstraction [5]. This pattern helps you structure a system into few layers and each site on top with each other. The well-know example of it is OSI 7-Layer model. It is often used in large and complex systems because it needs to be decomposed for ease the implementation. The advantage of using this pattern is it enhanced maintainability and extensibility and the drawback is it is expensive on system resources.

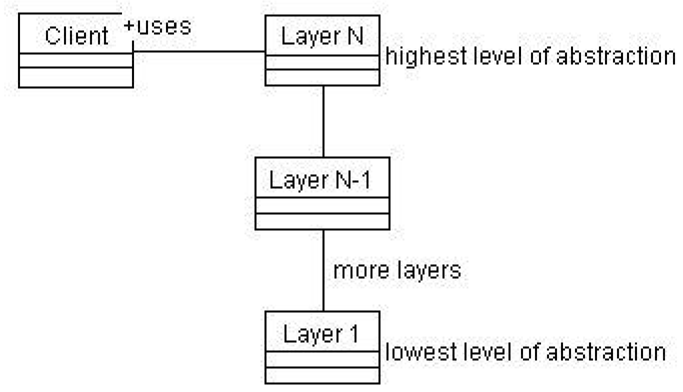

   Diagram of Layers Patterns [5]

2. Blackboard Patterns
   The Blackboard pattern combines several specialized subsystem and utilize their knowledge to provide a solution. This pattern is suitable for problem, which do not have the deterministic solution strategies [5]. The architecture of this pattern is a collection of components or subsystem, which are not interrelated to each other, but they share a common data structure. It is more efficiency but it may be more complex for error handling. A well-known example for this pattern is HEARSAY-II speech recognition system [5].

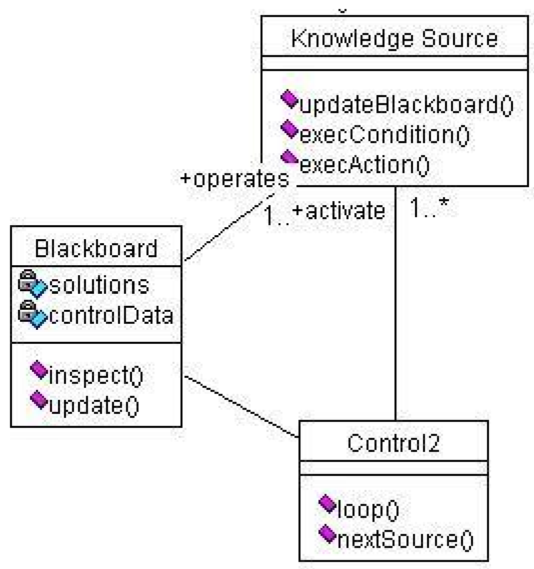

Diagram of blackboard patterns [5]

3. Pipes and Filters Patterns
   The pipes and filters pattern provides a structure for systems that process a stream of data [5]. The architecture of pipes and filters patters uses several sequential processing step/pipes to process data. Each pipe process contains a filter component. Data passes through pipe and filter. It is faster to build as compared to the above two patterns but it will create more data transformation overhead and share the state information is expensive and inflexible. This pattern is suitable for parsing a data stream such as XML.

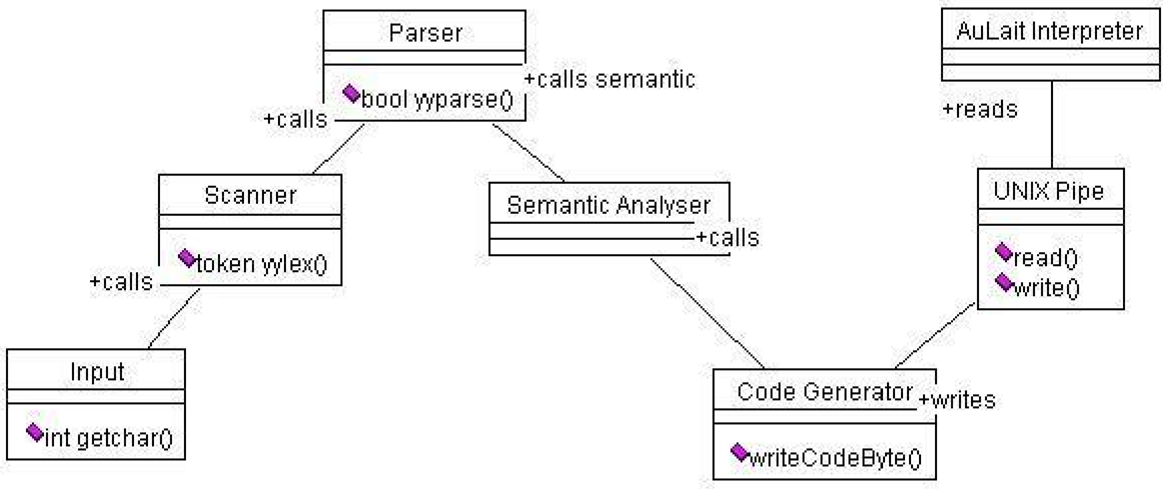

Diagram of Pipes and filters patterns [5]

Distributed Systems
Provides a complete infrastructure for distributed applications. Broker is the only pattern in this category. Pipes and filter pattern and microkernel pattern are also fall into this category. However, they are discussed in other categories since they are more suitable to other category.

1. Broker Pattern
   The broker pattern mainly uses in distributed system. It can be used to structure software systems with decoupled components that interact by remote service

invocation. This pattern can achieve better decoupling of clients and servers. Although it is hard to debug and test, it provides many advantages such as it improve changeability, extensibility of clients and servers and provide location transparency. This pattern was used to specify the Common Object Request Broker Architecture (CORBA).

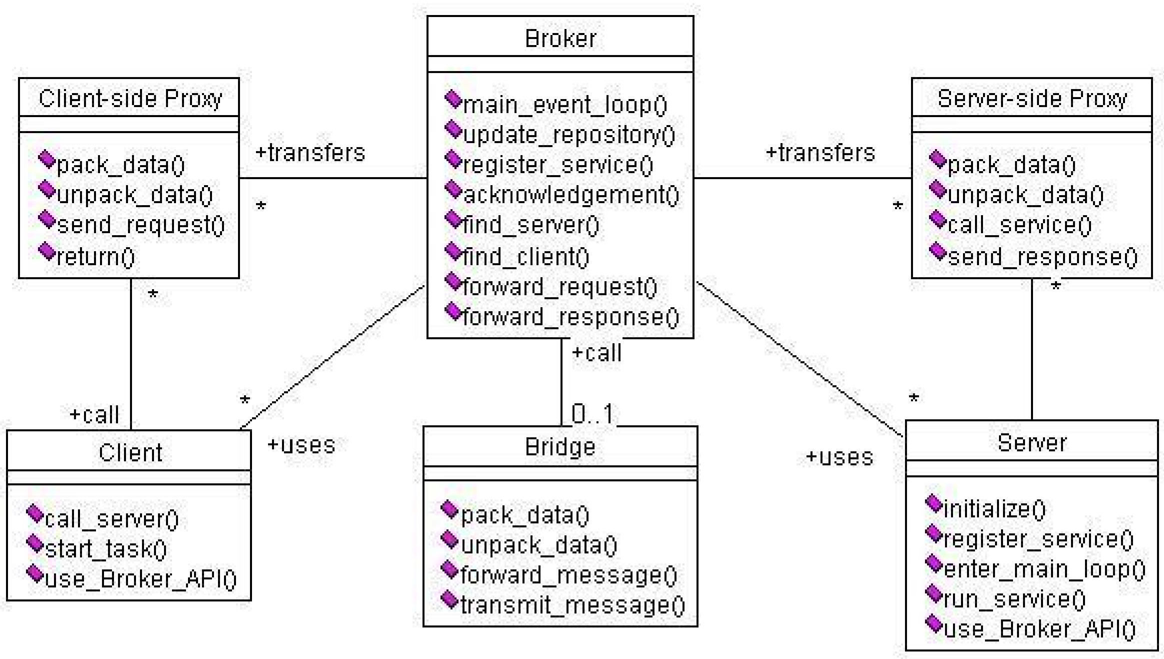

Diagram of Broker Pattern [5]

Interactive Systems
Interactive Systems support the structuring of software systems that feature human-computer interaction. This category of architecture pattern concentrate on handling user interaction. Model-View-Controller patterns and Presentation-Abstraction-Control patterns are the two patterns fall in this category.

Model-View-Controller Pattern
The Model-View-Controller pattern divides a system into three parts, which are model, view, and controller. The model describes how the whole system works. The view represents the view of the model. And the controller provides interactive between the user and the system. This pattern enhances maintainability and extensibility, as the same time, they are more complex to design. The well-known example can be found in Smalltalk architecture.

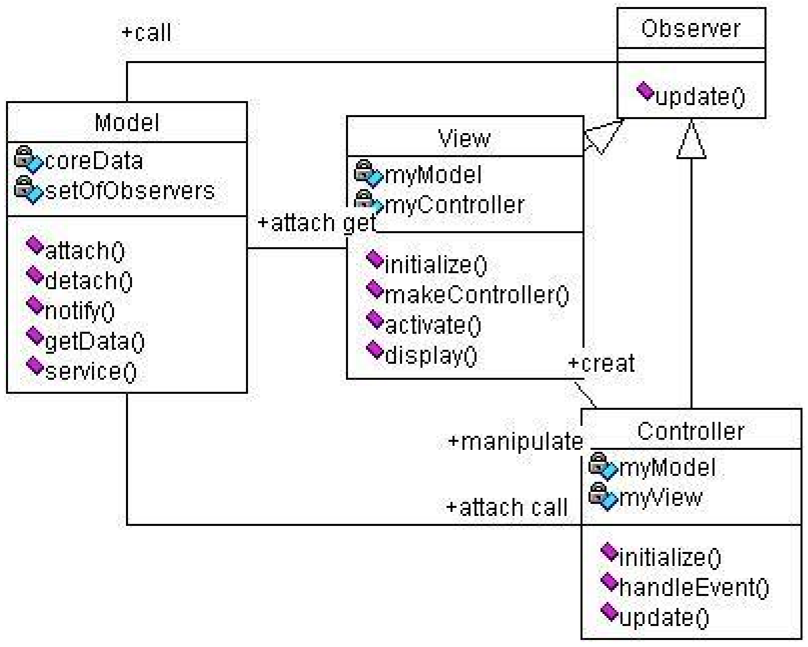

Diagram of MVC patterns [5]

2. Presentation-Abstraction-Control pattern
The presentation-Abstraction-Control pattern defines a structure for interactive software systems in the form of hierarchy of cooperating agent [5]. Every agent is responsible for a specific aspect of the application's functionality and consists of three components: presentation, abstraction, and control. This subdivision separates the human-computer interaction aspects of the agent from its functional core and its communication with other agents.

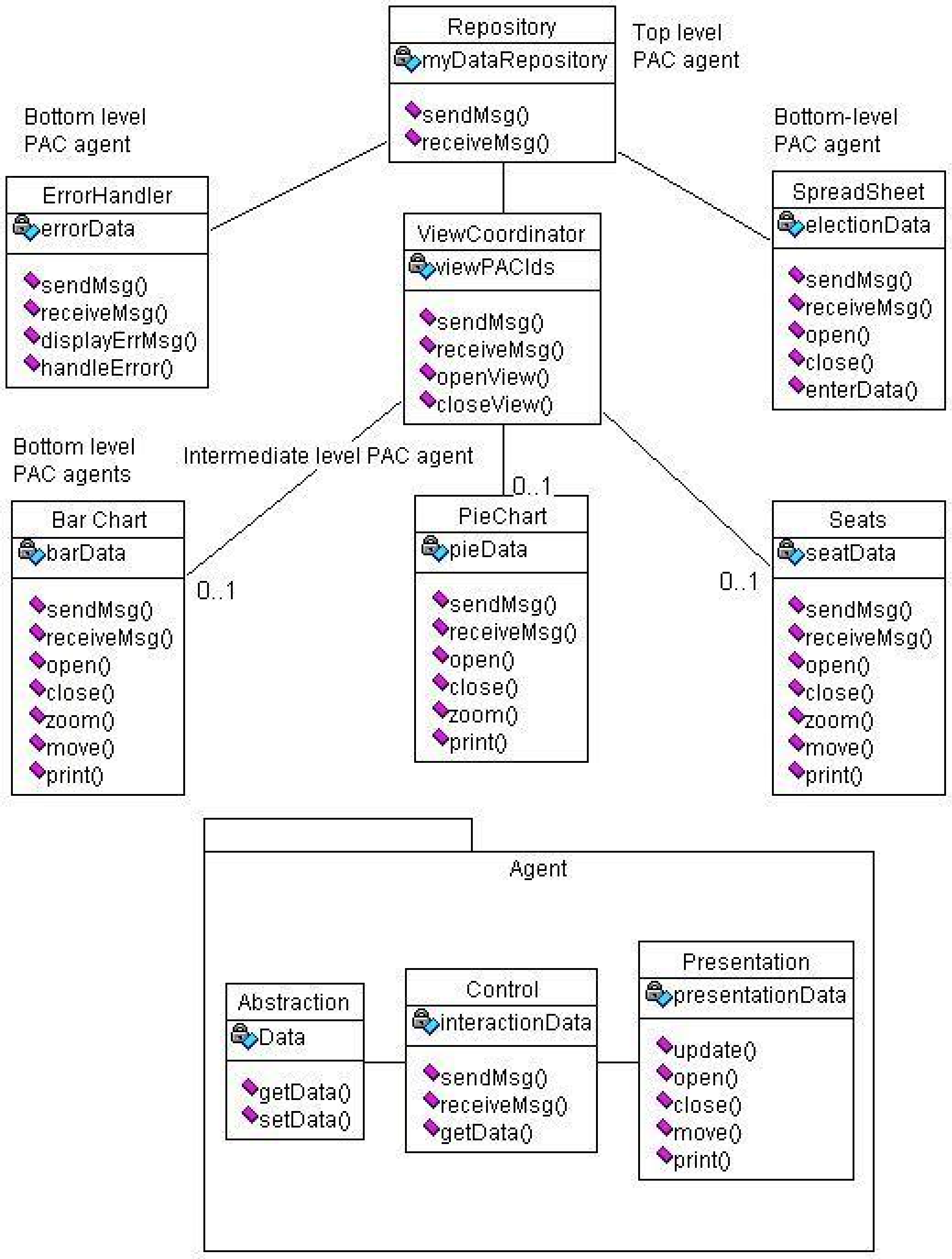

Diagram of Presentation-Abstraction-Control pattern

Adaptable Systems

A system will evolve over time; it changes its design or its functionality. Adaptable systems overcome this problem by supporting the extension of applications and their adjustment at the developed technology and change of the functional requirements. The two main patterns in this category are Microkernel pattern and Reflection pattern.

1. Microkernel Pattern
   The Microkernel pattern separates a minimal functional core from the extended functionality and customer-specific parts. It also serves as a socket of plugging in

these extensions and coordinating their collaboration. It allows component run on different process to communicate with each other. The mach operating system adopts this architectural pattern, which is developed at Carnegie-Mellon University. A commercial system called Chorus also uses this pattern.

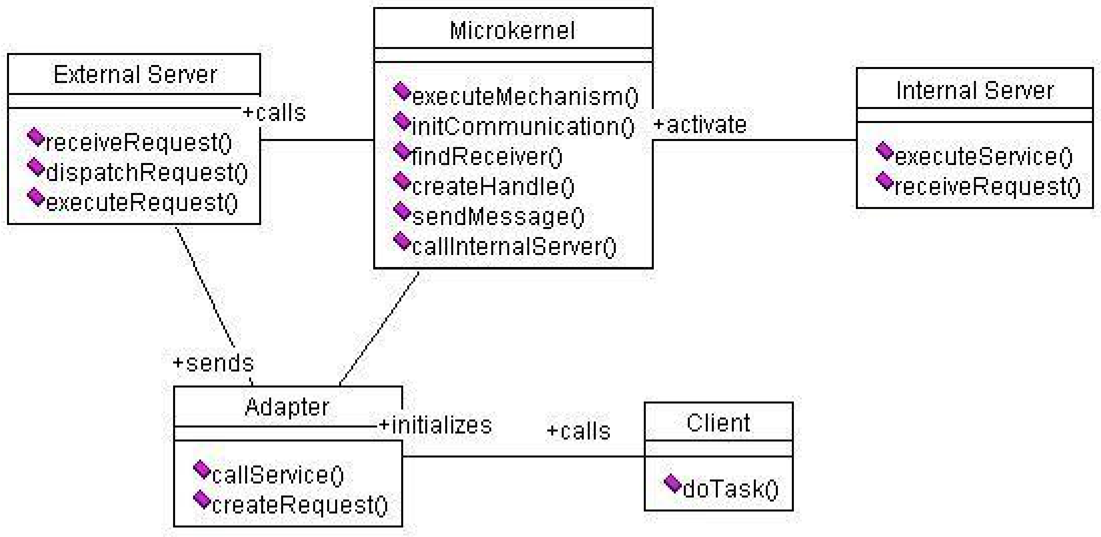

Diagram of Microkernel pattern [5]

2. Reflection Pattern
The reflection pattern provides a mechanism for changing structure and behaviors of software system dynamically. It support modifying the type structure, date element and function call mechanisms. In this pattern, an application is separate into two parts, i.e., a Meta level and a base level. A Meta level makes the software self-aware and provides information about selected system properties. The base level is in charge of the application logic. The implementation is kept in the Meta level.

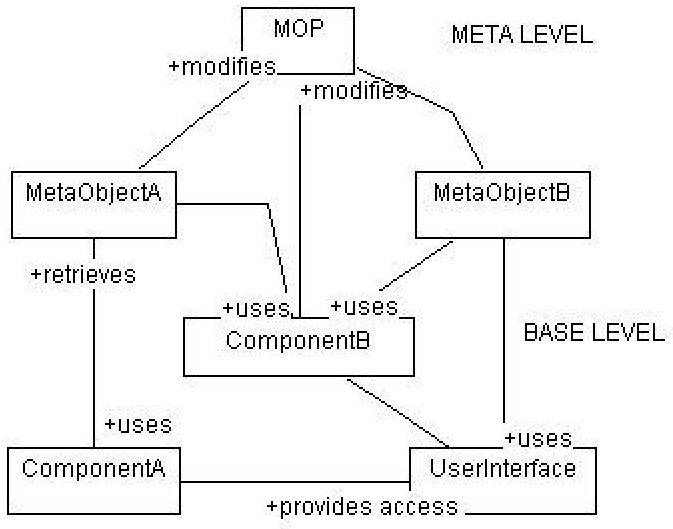

Diagram of reflection patterns [5]

Conclusion

This paper presented an overview of pattern-oriented software architecture. It first introduced software architecture, pattern language, and pattern-oriented software architecture. The second part of the paper presented the most common architectural patterns in the software pattern community. The architectural patterns, which discussed in this paper, have been used in many software projects and it has been proved. Therefore, system architecture should understand what architectural patterns are available and its usage. However, when designing the architecture of a system, human and environment factor is also play an important role. Such as company policy and resources available for the project. Project on time, under budget and the beauty of its architecture is determined by the architect, not by the architectural pattern s/he used.

Reference:

[1] Bass, Clements, and Kazman. *“Software Architecture in Practice”*, Addison-Wesley 1997:

[2] Alexander, Christopher, et al. *“A Pattern Language”*. New York: Oxford University Press, ©1977.

[3] Brooks, Frederick P., *"No Silver Bullet: Essence and Accidents of Software Engineering,"* Computer, Vol. 20, No. 4 (April 1987) pp. 10-19

[4] David Garlan. *“Formal approaches to software architecture”*. In David Alex Lamb and Sandra Crocker, eds., Proceedings of the workshop on Studies of Software Design, no. ISSN-0836-0227-93-352, in External Technical Report. Queen’s University Department of Computing and Information Science, May 1993.

[5] Buschmann F., Meunier R., Rohnert H., Sonnerld P., Stal M., *“Pattern-Oriented Software Architecture”*, John Wiley & Sons ISBN 0-471-95869-7, 1996

[6] ShawM., Garlan D., *“Software Architecture perspectives on an emerging discipline”* Prentice Hall 1996

[7] Pressman S. R., *“Software Engineering – A Practitioner’s Approach”*, fifth eds, McGraw-Hill 2001

[8] James O. Coplien, *“Software Patterns”*, SIGS Books & Multimedia, 1996