Component-based Architecture Reconstruction by Patterns

Shahrouz Moavena[*], Jafar Habibib, Alireza Parvizi mosaedc and Razie Alidoostid
[a, b, c, d,] Department of Software Engineering, Faculty of Computer Engineering, Sharif University of Technology, Tehran, Iran

Summery
nowadays, software architecture has obtained such a significant role as a foundation in software projects and processes that promoting objectives of software teams and organizations without it is impossible. Therefore, recognizing legacy architectures, processes, and systems, and recovering them for exploiting the benefits of clear and correct software architecture is very important; additionally, exploitation and analysis tools and expert systems should be used to construct architecture in the best way. In this paper, a method is defined which not only uses all existing information to reconstruct software architecture, but also makes use of structural, behavioral and semantic patterns in order to exploit their benefits in reconstructing architecture with better adaptability and compatible with quality attributes. The method by presenting some mechanisms and guidelines that consider patterns, provides users the ability of using it in different domains. Furthermore, the knowledge acquired from different abstraction levels are collected and used to establish knowledge trees and knowledge packages.

Keywords
Component, Software Reconstruction, Software Architecture, Architectural Style, Quality Attribute, Knowledge Management

1. Introduction

Software architecture, as a basis for software system development, represents a high-level abstraction of the system and performs a significant role by acting as a glue between every part of the software system [11].Software architecture not only is applicable in software projects and systems, but also is useful in and has important effects on processes and organizations.

These roles have become more considerable and efficient due to the today's complexity of software systems, projects, and processes and enlargement of organizations. Moreover, software architecture is an important tool for satisfying quality attributes and achieving desired quality for which there exist some architecture evaluation methods that can be considered as important software quality assurance techniques[4].

Documenting all architectural information during the design phase of the software system and updating it in order to reflect the current architecture of the system, is an important activity that must be performed during the software system development process in order to obtain mentioned objectives.

However, in the real world, architectural information becomes out-of-date and doesn't reflect the current architecture of the system [34]. This happens because the changes occur in the software system after development or even during implementation; which can happens intentionally or unintentionally.

Changes to a software system during implementation and maintenance can cause the architecture of a system to deviate from its documented architecture [13]. Hence, the difference between the conceptual architecture and the implemented architecture occurs which can effect on maintenance phase, even on the feedback from implementation to change the design base on a fault in the implementation [29].

The process of architecture reconstruction is a general process which covers a wide range of domains in which architecture has an important role and can effect on the quality of activity ; it may be a project, system, process or even organizations. In this process, experiences obtained from the past must be considered in order to achieve high quality results. Additionally, discovering and deciding about the hidden architectural styles and the relation among them, regardless of being homogenous or heterogeneous, is an important issue in the software architecture reconstruction which demands to be solved.

Several works have been performed in the architecture reconstruction field and several tools and techniques exist for identification and reconstruction of design patterns which are fine-granular and near to the code; however they can be improved and extended too. But to identify high-level abstract structures of system, process, or organization and reconstruct their architecture, extracting patterns form code cannot solve the problem.

This problem, as is considered in the paper, needs collection and integration of high-level abstract information of the system and can be solved by taking advantage of our previous works and integrating them with Formal Concept Analysis (FCA) [1] to create a complete process to cover the reconstruction problem. Additionally, process, architectural, design, and managerial patterns are exploited in the solution.

The important point that should be mentioned here is that making use of either tools or human knowledge at each levels of abstraction of the system will result in the acquisition of new knowledge. The newly obtained knowledge must be collected to establish new knowledge

Manuscript received July 5, 2016
Manuscript revised July 20, 2016

packages and knowledge trees which can be used in future decisions.

The remainder of this paper is organized as follows. In section two some related work is represented and discussed briefly. The important roles that software architecture performs in today's systems are included in the third section. The fourth section contains an introduction to architecture reconstruction and some related concepts. Our proposed method is represented in section six followed by a case study and the conclusion in two last sections.

2. Related Work

In the context of pattern recognition a lot of researches have been done, many valuable works have been carried out, and a lot of tools have been developed. However, none of them are general enough as they have a problem with higher levels of granularity and abstract levels. Discovering the relation between patterns and the way they interact, is another drawback of the existing methods.

Extracting information from the source code is one of the first steps in an architecture reconstruction process. The basic idea in some works lies in taking advantage of call-graphs. Generally, in software reverse engineering, call-graphs are used to understand the program and visualize behavior and structure of the code at different abstraction levels. In the mentioned research, at first a call-graph is constructed in the software by considering its source code and then, the obtained result is compared with the expected call-graph of the system to identify the differences between existing and expected architectures. This is because software architecture affects quality attributes and reconstructing it requires more than program comprehension and code level analysis [12].

Imagix [14] is a commercial tool which works on the C, C++ and Java source code to reverse engineer and analyze the code. By analyzing the data flow, the calculation trees are constructed which present information about the assignments leading to the current value of a variable. Moreover, they track assignment dependencies across functional boundaries and through parameter passing.

In [31], by considering the source code, an initial graph is established; this is done by taking advantage of a compiler. The graph is the basis for creating another graph which describes the design recovery process. Moreover, the validation is made with respect to some well-known design patterns such as Composite.

Moreover, some source code extraction tools exist for Java code. UFJ [15] is a high quality commercial tool that is used for reverse engineering of Java code and has an integrated static call graph viewer that depicts call dependencies between classes [12].

In [18], a tool is presented which is based on C++ code. The meta-information and patterns are extracted from C++ header files and are stored as Prolog rules.

In addition, in order to query, manipulate, and extract information about the source model automatically, some reverse engineering tools have been developed. In [13], a system, called IAPR, is described in which patterns are discovered within software architectures. This is done by implementing a heuristic form of sub-graph isomorphism and matching patterns to an architecture.

The method represented in [24], is used to extract source model from lexical specifications by generating small and easy-to-write specifications with few constraints and by considering almost all information around an artifact.

Another work, takes advantage of Labeled Transition System (LTS) to represent the software architecture behavior and exploit some architectural tests. The set of architectural test sequences is obtained by covering abstracted views of the LTS description of the software architecture behavior. The test sequences are then refined into concrete tests and to be executed on the implemented system [5]. Moreover, in order to reduce the possible sequences of transitions to a limited number of test sequences an observation function is used on the LTS. However, establishing a relationship that maps high-level test sequences on concrete and executable test cases is very complicated. The problem arises especially in the absence of a rigorous, formalized refinement process from the software architecture specification down to the source code[5].

In [32], a method has been proposed to determine software architecture and its shared and variable parts in software product line. The basic purpose of this method is an evaluation of the potential of establishing software product line; however it can be used for architecture reconstruction. The architectural model of the system is achieved by abstracting the implementation model extracted from the existing codes. The architectural styles and patterns are then discovered by adapting them with the architectural model.

The method presented in [26], is used for design recovery and understanding the program. Additionally, [26] provides a tool to semi-automatically recognize instances of design patterns and help to the presented method. It takes advantage of an incremental algorithm for which the related information such as the domain and context knowledge should be provided. In order to detect the patterns, at first a special form of ASG (annotated abstract syntax graph) is constructed and then a pattern neighborhood is defined with respect to the lattice by taking advantage a sub-graph matching algorithm.

ISA [27] is a tool for automatic qualitative analysis of software architecture in which architecture analysis and qualitative estimation of software are performed based on the recognition of patterns from architectural descriptions represented in the UML diagram.

In [17], a reverse engineering tool is represented which has useful features for capturing certain architectural views in UML notation. However it can only reverse engineer UML semantics in some prepared situations.

The environment presented in [16] is used for reverse engineering of design components based on the structural descriptions of design patterns.

The tool presented in [7] detects design patterns in Smalltalk environments by taking advantage of a cycle-detection technique. In order to encode patterns no general abstraction has proposed and a clearly generalized approach to detect patterns has not demonstrated.

A method to improve and query design patterns based on machine learning techniques has been proposed in [8]. This method, takes advantage of the Columbus framework for reverse engineering. It constructs architectural patterns based on code analysis and creation of an Abstract Semantic Graph; then the pattern description represented in DPML, is used for structural adaptation of the patterns. The weakness of Columbus is in determination of patterns which are similar with each other structurally. However, some efforts have been done to solve the problem somehow, by exploiting decision tree C4.5 [30] and neural networks [6].

In order to make tool interoperability easier, some frameworks have been proposed [34],[33]. In [33] the strategies for collapsing information while building abstractions during architecture reconstruction have been represented. Additionally, by identifying the situations in which multi-collapsing is required understanding a system or its particular aspects becomes easier. The reference framework proposed in [28], can be used in classifying and comparing existing techniques. Additionally, that makes it possible to discover the problems in software architecture reconstruction and find existing viewpoints.

In order to evaluate the accuracy of tools of reverse engineering which are applied in architectural recovery and address their usability a comparison has been performed in [2].

In [25], a high-level structure of the system is specified by the architect and then it must be mapped to the source code. An open source tool has been developed to support the method and specify the degree to which the high level model agrees with or differs with the source code.

The method proposed in [13], pays attention to the higher levels of abstraction in architecture reconstruction by identifying architectural patterns of an existing system. ARM, which stands for Architecture Reconstruction Method, is a semi-automatic analysis method and codifies heuristics for applying existing reverse engineering tools to the problem of recognizing more abstract patterns in the implementation [13]. However, the ARM can be criticized from different aspects. The two presented case studies are very simple and have a clear internal architecture; they are not complete and are not capable of identifying high level of the system. The proposed method has not extended after that, while they have acknowledged that the solution is applicable in just some specific problems.

Finally, we should mention two points. First, there are some tools in the literature that have been superseded or are only research prototypes[12]. Second, in a comprehensive solution in this domain the quality attributes and system and environmental specifications must be considered; like what is done in architecture design. To understand the problem domain and identify the suitable styles and patterns, several dimensions must be considered in which the code structure is just one of the important dimensions.

3. Software Architecture Roles

Software architecture is considered as a matured branch in the whole of the software engineering domain for which variety of techniques and methods are applied to promote. Software architecture works as a conceptual glue to connect different phases of a project or even parts of an organization or a process [11],[9]. Software architecture, in addition to specify connections and roles, provides a framework for communication of components and stakeholders and models the risks, projects, processes, and organizations in a suitable structure based on quality attributes [4].

The complexity of systems and structures, causes the architecture design and the reconstruction face with several problems. In this domain, like the other matured domains, some patterns created which are representing successful solutions [10]. Making use of architectural patterns and, following that, process patterns, which are originated from the structure and architecture of processes, is one of the most beneficial and most efficient solutions in the fields of reconstruction and even construction of architecture [10], [13]. Patterns perform a valuable role in the field of reconstruction because we can never completely identify architecture of legacy systems which are complex and widespread. But the solution of taking advantage of patterns and mapping the problem domain to them is very suitable because selecting and making use of patterns, even if has not done at the beginning of the project and design phase, is implementable in reconstruction and provides the maintenance team with high capabilities and capacities.

Software architecture and its classic patterns and styles are improving and advancing every day and constructing and using a composition of them is widespread. The performance and coverage of architecture styles in complex problems are unavoidable but bring us with a lot of problems itself. Reconstructing architecture by means of heterogeneous styles is considered as a valuable activity in this domain because it gives the development team high abilities and capabilities to model the system in the best way [12]. However, existence of identification mechanisms for discovering and identifying suitable composite patterns for each system or subsystem, needs tools, methods, and approaches which is possible just by integrating all the existing knowledge.

4. Architecture Reconstruction

Software architecture reconstruction, also mentioned as software architecture recovery, deals with the extraction and analysis of a system's architecture [13]. The origin of software architecture reconstruction is in software reverse engineering in which the focus is on understanding the program and visualizing the structure and behavior of code and acts at different levels of abstraction [12]. Retrieving a documented architecture for an existing system is one of the most important objectives of software architecture reconstruction which is categorized in reverse engineering activities [10].

Success of software architecture reconstruction activity highly depends on the identification and extraction of important and significant information related to the architecture with respect to different aspects. Although this can be performed somehow by considering documentations of the software product, but it is not available every time in every system and moreover quality of the achieved/recovered architecture depends extremely on the precision of existing documents; which in many cases have not appropriate accuracy. On the other hand, in most cases, the source code of the software is the only trustworthy available resource.

The need for architecture reconstruction appears when some changes are made to the architecture during software implementation or maintenance and no effort is made to maintain and update the architecture documents. Hence, the current architecture may drift from the expected, documented architecture. This will result in prevention of exploiting from architecture benefits in maintenance and making further changes to the system which demands a good understanding of the software architecture.

In the case of the need for changes in an existing system, and maintenance in general, existence of useful design documents is very important. Moreover, usefulness of design documents can help in easily evaluating the closeness and conformance between documents and the code.

at higher levels of abstraction, and in order to find the architecture of the system, code level analysis and program comprehension are not enough [22]; in this context, finding architectural styles and patterns and the relation among them are of high importance. However, in spite of several researches in the field of architecture reconstruction, there is a little research on developing effective and efficient methods for higher levels of abstraction in architecture recovery [13].

5. Architecture Reconstruction Processes

As mentioned in the related work, in order to extract design patterns from code, several advanced processes have been constructed and variety of tools have been developed that each one has a different proficiency with respect to the objective of its creation. But they are not even capable of identifying design patterns let alone performing action at a higher level to identify architectural styles and, generally, system architecture.

In this domain which is the main goal of this paper, much work has not been taken and the existing researches are not capable of being applied practically and even, in some cases, are not complete theoretically and cannot be represented or argued.

Generally, there exist two types of processes in architecture recognition. First are bottom up processes which start from code and low-level design and try to construct higher-level abstraction with respect to the low-level information. The other types of processes are top down and try to find a software architecture by constructing a graph of the system and then mapping between high-level model and low-level design.

In this paper not only a combination of the two mentioned methods are exploited in order to solve the problem, but also other advanced techniques, such as FCA, are used to identify heterogeneous compositions; the effort which seems impossible but the fact is that absence of integration of information, knowledge and existing technologies makes this problem very complicated. However, constructing a framework or a method which provides a discipline for collecting and integrating existing information and techniques can help in solving such problems.

A fundamental concept about successful architects is that they should have a widespread but shallow knowledge of different sciences and techniques and use them in architecture design. This means that when we want to perform an action in reverse, we should take advantage of the knowledge to obtain the basic model of the architecture and efficient reconstruction results. In this context, mapping simple solutions of other domains, such as data mining, expert systems, and neural networks, might help in solving the problem.

The solution proposed in this paper, is based on four main strategies. Additionally, by taking advantage of small subsidiary solutions and techniques, it uses the context information all together for identification and reconstruction. Also, by exploiting a knowledge management system, the knowledge acquired in different levels can be stored and classified to construct a knowledge tree for software architecture.

As mentioned, the proposed process consists of four types of main components. First, are feature extraction tools; these tools must be capable of extracting functional and non-functional requirements of the system. Second, is making use of FCA to identify the collaboration among patterns. Third, is constructing a hierarchical framework for knowledge management. And fourth, is making use of a description language which supports heterogeneous architectural styles and has semantics in it.

Figure1 illustrate, at a high level view of reconstruction. It is an extension of previous work of Decision Support System Framework[20]. It uses the two layers of Component Control which collaborates with concrete components and Architecture Management which analyzes feature and produces a new plan for reconstructing architecture of the system. FCA component formalizes relation of styles when they have been described by standard language. So Architectural styles must be modeled by the Description Language.

Finding the collaboration among different parts of the system is an important phase in our approach. In order to achieve the goal, we take advantage of FCA [1], which is used for detecting classical design patterns.

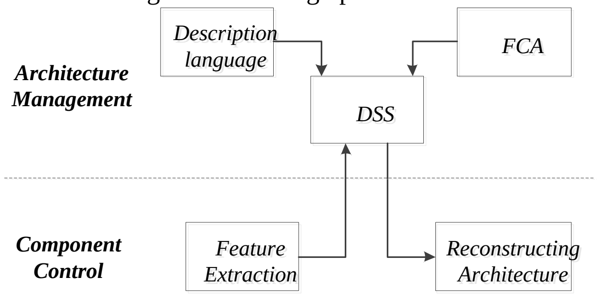


Fig 1. high level view of the Reconstruction Framework

FCA, by taking advantage of lattice theory, identifies meaningful groupings of objects that have common attributes. We will show how taking advantage of FCA can help us in our hierarchical structure [22] to find the relation between architectural patterns and the way they interact.

In order to use FCA, the elements and properties of a context should be defined. Elements are tuples of classes from the analyzed application and properties are relations inside one class tuple. After that, groupings based on the common properties of the elements, named as concepts, are performed. The set constitutes a concept lattice by taking advantage of some algorithms [1]. For example Fig 2 is a sample class diagram and Fig 3 is its lattice [1].

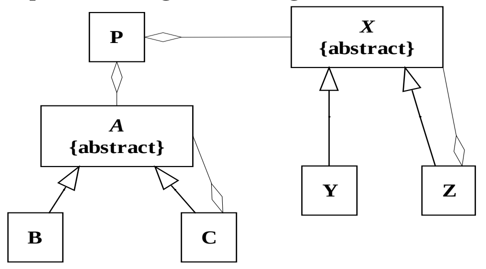


Fig 2. Example class diagram[1]

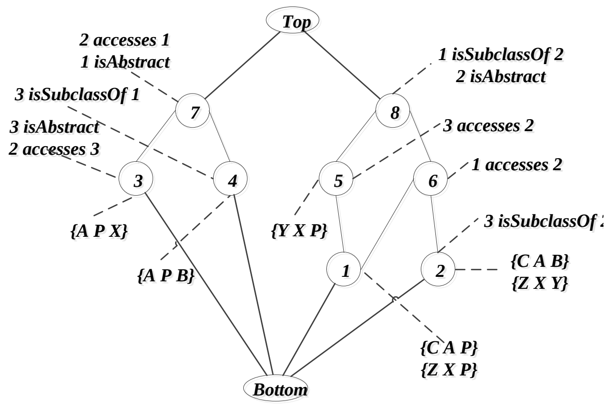
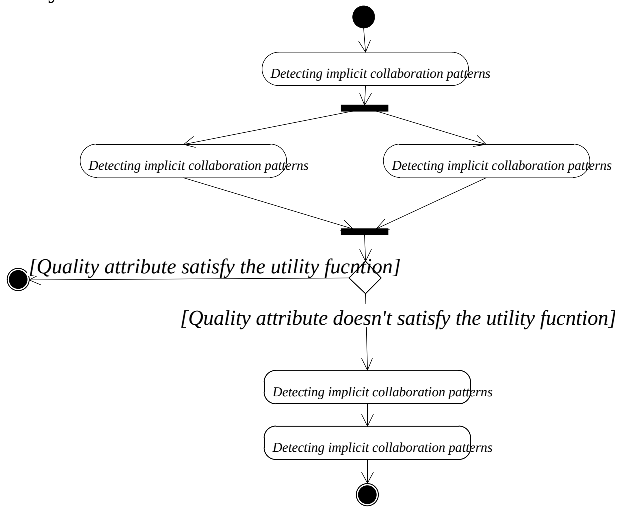


Fig 3. Lattice of Fig 2 [1]

Any node of Lattice equal to a pattern and notes on node represents some features of the pattern. For instance, node8 is a pattern that its instances are members of {{C A P}, {Z X P}, {C A B}, {Z X Y}, {Y X P}} such that second class is subset of first class and second class is an abstract class in any member of the set. In addition, C pattern is accessible by B pattern when features of B are a subset of features of C[1].

As mentioned later, there are two up-down and bottom-up reconstruction process which their activity diagrams are illustrated in the Fig 4 quality attributes and functional requirement extracted from source code by Feature Extraction component.

The Component Control layer fetches collaboration patterns and compare them with navigation patterns such as Façade that is stored in the repository. Any collaboration patterns match to a navigation pattern by some method such as neighborhood analysis, mining pattern and identify coding styles[1]. In result, navigation pattern is used to measure the quality attributes of the system. In continue, this layer compares the quality attribute with threshold function which expert system proposed and produce a reconstruction plan when the quality attributes do not satisfy the threshold function.



Detecting implicit collaboration patterns
Detecting implicit collaboration patterns
Detecting implicit collaboration patterns
[Quality attribute satisfy the utility fucntion]
[Quality attribute doesn't satisfy the utility fucntion]
Detecting implicit collaboration patterns
Detecting implicit collaboration patterns

Fig 4. bottom-up architecture reconstruction process

Style composition is a useful method for creating the reconstruction plan. In order to construct a hierarchical composition style, we use the tree structure defined in [3]. In this structure, the nodes which are near the root, i.e, exist in lower depths, are more general and might be heterogeneous architectural styles. When we go through the depth of the tree, each heterogeneous architectural style is decomposed to its constructors; it is represented in Fig 5.
This figure represents that some styles such as style 2.1 and 2.2 can be embedded in another style such as style 2. Quality of embedded style effects on the quality of its parent with a weight named OD such as performance of style 2.1 effects on performance of style2 with weight of OD2.1.

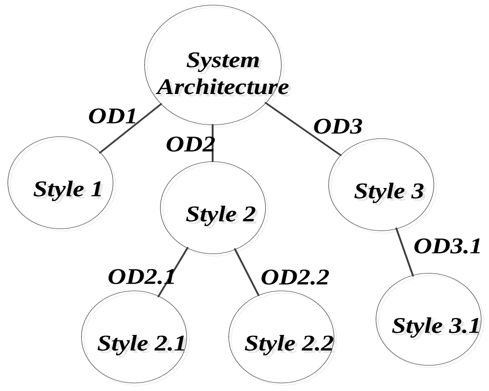


Fig 5. A view of hierarchical structure in the middle levels, contains heterogeneous styles [3]

It is obvious that quality attributes are leafs of the tree, as shown In Fig 6. To construct the tree, we use a bottom-up fashion in this approach. By identifying the quality attributes, the roots of the tree have achieved. Moreover, we made use of FCA in the previous step to find the relation among quality attributes and grouping them to detect design patterns, but we cannot detect the architectural patterns in the FCA. However, the tree can be refined in this step.

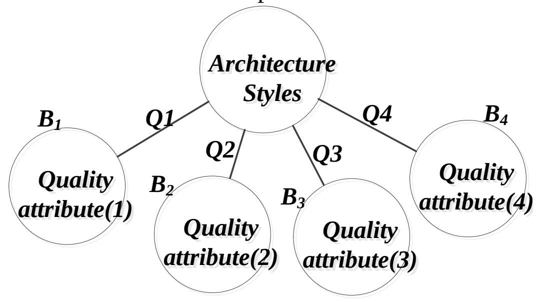
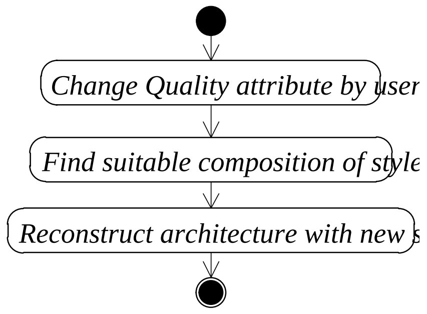


Fig 6. A view of hierarchical structure of a style [3]

Finally, Reconstructing Component deploys the created style in the software architecture.
Fig 7 illustrates the top-down reconstruction process which starts with a user query for changing the quality attribute. Architecture Management layer creates a new style and reconstruct the architecture as mentioned in bottom-up process.



Change Quality attribute by user
Find suitable composition of style
Reconstruct architecture with new s

Fig 7. top-down architecture reconstruction process

By coming back to the related work section, we understand that Imagix and RMTool are two useful tools that can be used for feature extraction. Imagix has a bottom-up viewpoint while RMTool works top-down. By taking advantage of these two types of approaches, extracting features of a system will be easier as each one can completes the other and their results can be used as valuable information for each other and for taking the final decision.
The tool developed in [20] to support a framework for capturing and using architectural knowledge to improve the architecture process can be used for architecture reconstruction. The tool has a repository in which general scenarios are stored and has an interface for capturing them. The knowledge which exists in this tool can be integrated

into packages in order to construct a knowledge tree. Moreover, the knowledge acquired while constructing the tree, can be added to the tool.

However, in order to detect architecture patterns, we need a framework capable of making decision based on all of the existing knowledge. In this context, we can take advantage of our decision support system presented in [19],[21],[23]. In order to make appropriate and precise decisions, we need to prepare all exiting information for the deciding environment. The knowledge exists in the knowledge management tool, the knowledge of the expert architect, the relations between discovered attributes and the knowledge of the context are important inputs of the decision support system.

In the process of recovering software architecture an important aspect is highlighted which is the semantics and concepts of the architecture; from this perspective, each architectural style and pattern, in order to be constructed, has some conceptual reasons behind its existential philosophy which can act as a very powerful recognition factor. Several efforts have been taken in this area; not for recognizing architecture based on semantic but in order to make description languages capable of representing semantic.

In the definition of each architectural style, Pahl and his colleagues [28] in their laboratory proposed a modeling language. The modeling language is used to integrate styles, structures and behaviors into a coherent framework and understand the relation between quality requirements and conceptual architecture styles. By using this modeling language, an architecture style ontology is defined which is consist of components, connectors, roles, ports, and configuration. In order to define an architectural style, at first some basic notations must be defined.

6. Case Study

The composed method represented in this paper, which has been designed in a pattern-oriented process and based on architectural patterns, is used in the FlashDevelop project. FlashDevelop is an open source project which created by passionate Flash developers for flash developers. It is a .net web development IDE for compiling and generating code, project compilation and debugging and etc[35].

RMTool and Imagix, which represented in the related work section, are used to extract the pattern from FlashDevelop. Because of the limitation in time and some technical error in mapping the FCA on it, the result of that is not represented in this paper but the primary result was good enough to understand the benefits and weaknesses of our approach. We defined some relation among patterns such as include, sequential, parallel and etc, and a utility function for performance. This utility function must be dependent on pattern relationship and is defined by expert users. our utility function was longer than this paper, thus we exhibit some part of our utility function pseudo code.

```
Function UtilityFunction(get a set of patterns)
Begin
    For any A and B patterns in set
      If A is subset of B then evaluate F1 function and add to
    UtilityFunction value
      If A is parallel with B then evaluate F2 function and add to
    UtilityFunction value
    <and other relationship>
end
```

Also we modeled primary design pattern, architectural styles and relation among them with FCA and stored in the repository. In the result any pattern is stored as same as the graph is mentioned later. Also we developed a program with C++ that searched the styles and created a tree of styles with an evolutionary algorithm that its objective function is the utility function.

To continue, due to design patterns of the repository had 3 or 4 elements, we filtered RMTool to extract only patterns with 3 or 4 elements. Then we executed the RMTool on the FlashDevelop and extracted design patterns and measured the performance. Utility function got design patterns and their relationships and measured the performance. The result showed that quality of the architecture is satisfied because performance was 21 whereas the threshold was 18. Table 1 is some part of extracted design patterns and illustrate that number of patterns with high performance such as singleton, composition, the iterator is more than patterns with low performance such as Factory Method, Façade, Mediator.

Table 1. low and high performance patterns in FlashDevelop

high performance patterns		low performance patterns	
Pattern	number	pattern	number
Singleton	205	Factory Method	32
Composition	167	Façade	56
Iterator	51	mediator	41

So, we added some interface between components purposely and executed RMTool. The Result, Table 2, shows that number of Façade, client-serve and proxy patterns are increased.

Table 2. low and high performance patterns in changed sample

high performance patterns		low performance patterns	
Pattern	number	pattern	number
Singleton	205	Factory Method	45
Composition	120	Façade	170
Iterator	51	mediator	41

As we expected, the value of performance (by the value 12) was lower that threshold performance (by the value 18). So performance did not satisfied utility function.
DSS component searched in the style tree with our heuristic algorithm and proposed some suitable composite styles which satisfied utility function. We selected best composite style which is illustrated in Fig 8 and is composition of the Blacklist, Pipe-Filter and call-Rerun styles with a depth of three. Note that depth of style tree must be lower than 6 in our approach because we propose that depth 6 is threshold between architectural styles and design patterns.

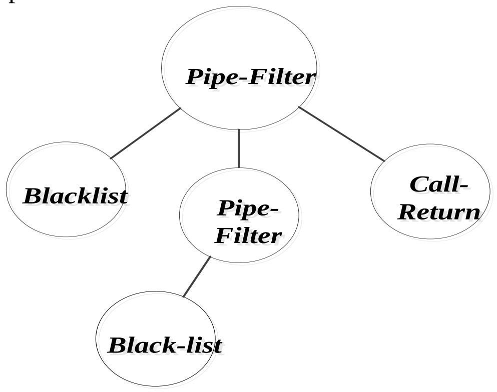


Fig 8. selected pattern composition

Finally, results of deploying selected style on FlashDevelop system represent that performance is 19. It means, performance was satisfied. In this case study we store 30 design patterns and 10 architectural styles in the repository due to we want to represent our reconstruction approach but we can increase the accuracy of our approach by automatic modeling and storing runtime detected patterns.

7. Conclusion

Quality of software is a very important aspect of software whereas maintenance of software quality is noted as a repetitive activity after software realization. Architecture reconstruction is a useful method for resolving this problem. The architecture is reconstructed in order to exploit useful specifications which come with the structural and documented architecture. These specifications are highlighted when using architectural styles and patterns; hence, an impossible process is made possible.
In this paper, generally, taking advantage of pattern, either in architecture or in process or in knowledge management, has been considered. The defined structure, by integrating behavioral information of different application domains and taking advantage of some techniques, provides the ability of detecting and discovering heterogeneous structures. The technology mapped in this domain (FCA) provides us with the ability to reconstruct software architecture by considering the tree structure constructed for architecture styles in line with the information obtained from different parts.
As future work, the proposed processes will be applied in a real project which creates an efficient practical evaluation and makes us capable of completing processes with respect to the obtained experiences.

References

[1] ARÉVALO, G., BUCHLI, F., and NIERSTRASZ, O., 2004. Detecting implicit collaboration patterns IEEE, 122-131.
[2] ARMSTRONG, M.N. and TRUDEAU, C., 1998. Evaluating architectural extractors IEEE, 30-39
[3] BABAR, M.A. and GORTON, I., 2007. A tool for managing software architecture knowledge IEEE, 11-11
[4] BASS, L., CLEMENTS, P., and KAZMAN, R., 2003. Software architecture in practice. Addison-Wesley Professional.
[5] BERTOLINO, A., INVERARDI, P., and MUCCINI, H., 2003. Formal methods in testing software architectures. Formal methods for software architectures, 122-147.
[6] BISHOP, C.M., 1995. Neural networks for pattern recognition.
[7] BROWN, K.G., 1996. Design reverse-engineering and automated design pattern detection in Smalltalk North Carolina State University.
[8] FERENC, R., BESZÉDES, Á., TARKIAINEN, M., and GYIMÓTHY, T., 2002. Columbus-reverse engineering tool and schema for C++ IEEE, 172-181.
[9] FIRESMITH, D.G., CAPELL, P., HAMMONS, C.B., LATIMER, D., and MERENDINO, T., 2008. The method framework for engineering system architectures. Auerbach Publications.
[10] GARLAN, D., 1996. Style-based refinement for software architecture ACM, 72-75
[11] GARLAN, D., 2000. Software architecture: a roadmap ACM, 91-101
[12] GORTON, I. and ZHU, L., 2005. Tool support for just-in-time architecture reconstruction and evaluation: an experience report IEEE, 514-523.
[13] GUO, G.Y., ATLEE, J.M., and KAZMAN, R., 1999. A software architecture reconstruction method Kluwer, BV, 15-34.
[14] HTTP://WWW.IMAGIX.COM, 2013.
[15] KAASTRA, M.D.A.K., C.J, 2003. Toward a semantically complete Java fact extractor. In Department of Computer Science, University of Waterloo.
[16] KELLER, R.K., SCHAUER, R., ROBITAILLE, S., and PAGÉ, P., 1999. Pattern-based reverse-engineering of design components IEEE, 226-235
[17] KOLLMANN, R., SELONEN, P., STROULIA, E., SYSTA, T., and ZUNDORF, A., 2002. A study on the current state of the art in tool-supported UML-based static reverse engineering IEEE, 22-32

[18] KRAMER, C. and PRECHELT, L., 1996. Design recovery by automated search for structural design patterns in object-oriented software IEEE, 208-215.
[19] MOAVEN, S., HABIBI, J., AHMADI, H., and KAMANDI, A., 2009. Decision Support System Environment for Software Architecture Style Selection. In Proceedings of the 21th conference on Software Engineering and Knowledge Engineering SEKE'09 (Boston, USA2009), Knowledge System Institute, 147-151.
[20] MOAVEN, S., HABIBI, J., AHMADI, H., and KAMANDI, A., 2008. A decision support system for software architecture-style selection IEEE, 213-220
[21] MOAVEN, S., HABIBI, J., AHMADI, H., and KAMANDI, A., 2008. A Fuzzy Model for Solving Architecture Styles Selection Multi-Criteria Problem IEEE, 388-393
[22] MOAVEN, S., KAMANDI, A., HABIBI, J., and AHMADI, H., 2009. Toward a Framework for Evaluating Heterogeneous Architecture Styles IEEE, 155-160.
[23] MÜLLER, H.A., JAHNKE, J.H., SMITH, D.B., STOREY, M.-A., TILLEY, S.R., and WONG, K., 2000. Reverse engineering: a roadmap ACM, 47-60
[24] MURPHY, G.C. and NOTKIN, D., 1996. Lightweight lexical source model extraction. ACM Transactions on Software Engineering and Methodology (TOSEM) 5, 3, 262-292.
[25] MURPHY, G.C., NOTKIN, D., and SULLIVAN, K., 1995. Software reflexion models: Bridging the gap between source and high-level models ACM, 18-28
[26] NIERE, J., SCHÄFER, W., WADSACK, J.P., WENDEHALS, L., and WELSH, J., 2002. Towards pattern-based design recovery ACM, 338-348
[27] PAAKKI, J., KARHINEN, A., GUSTAFSSON, J., NENONEN, L., and VERKAMO, A.I., 2000. Software metrics by architectural pattern mining, 325-332.
[28] PAHL, C., GIESECKE, S., and HASSELBRING, W., 2007. An ontology-based approach for modelling architectural styles. Software Architecture, 60-75.
[29] POLLET, D., DUCASSE, S., POYET, L., ALLOUI, I., CIMPAN, S., and VERJUS, H., 2007. Towards a process-oriented software architecture reconstruction taxonomy IEEE, 137-148
[30] QUINLAN, J.R., 1993. C4. 5: programs for machine learning. Morgan kaufmann.
[31] SEEMANN, J. and VON GUDENBERG, J.W., 1998. Pattern-based design recovery of Java software ACM, 10-16.
[32] STOERMER, C. and O'BRIEN, L., 2001. MAP-mining architectures for product line evaluations IEEE, 35-44
[33] STOERMER, C., O'BRIEN, L., and VERHOEF, C., 2004. Architectural views through collapsing strategies IEEE, 100-110
[34] VAN DEURSEN, A., HOFMEISTER, C., KOSCHKE, R., MOONEN, L., and RIVA, C., 2004. Symphony: View-driven software architecture reconstruction IEEE, 122-132
[35] http://www.flashdevelop.org/, 2013.