Journal of Computing and Security

April 2014, Volume 1, Number 2 (pp. 83–93)

http://www.jcomsec.org

Ranking of Common Architectural Styles Based on Availability, Security and Performance Quality Attributes

Safieh Tahmasebipour [a]
Seyed Morteza Babamir [a,*]
[a] *Department of Computer Engineering, University of Kashan, Kashan, Iran.*

ARTICLE INFO.

Article history:
Received: 27 December 2012
Revised: 31 August 2013
Accepted: 27 October 2013
Published Online: 5 September 2014

Keywords:
Software Architecture, Quality Attributes, Availability, Security, Performance.

ABSTRACT

An architect designs software using architectural styles and quality attributes. To satisfy quality attributes, architectural tactics are used. An architectural tactic determines how to implement a quality attribute in architectural styles. The interaction between architectural tactics and architectural styles plays an important role in meeting quality attributes. In this article, the interaction between architectural tactics, namely “availability” and “performance”, and architectural styles, namely “pipe & filter”, “layered”, “blackboard”, “client-server” and “broker” is evaluated, and then, a new ranking scheme for the architectural styles is proposed. Through this scheme, the best architectural style maybe obtained for every individual quality attribute or their combinations.

1 Introduction

From different point of view, several definitions has been proposed for the concept of architectural style [1, 2]. An architectural style indicates the structure and common properties of a software family rather than single software. Moreover, a style is a set of components with connecting connectors and constraints that govern the connections. Such a set forms a configuration that is governed by some constraints.

Non-functional (quality) requirements play a critical role in software development and are used as a selection criteria for choosing a proper design and ultimate implementation, [3]. In fact, they are software properties and qualities which should be satisfied along with functional requirements or services [4]. The quality attributes affect software final operation, satisfaction of quality attributes and special structures that should be used in software production process. In this study, the quality attributes of security, availability and performance are considered. A tactic is a design decision that affects the realization of a quality attribute, [5], [6] and [7]. This means that implementation of an architectural tactic results in meeting a special quality attribute of an intended architectural style. As shown in Figure 1, architectural tactics are intersection point of architectural styles and quality attributes. The redundancy tactic for instance, is used to promote the availability quality attribute. A tactic may require essential changes in the structure and behavior of style or it may need a new structure and behaviour. Tactics may be determined at software design or software run time. The information hiding tactic, for example, is determined at software design time to improve the modifiability property while the authenticity tactic is a run-time tactic enhancing the security quality attribute. In this study, the concern is the run time tactics inasmuch as: (1) we are inspired by [8], knowing that run-time tactics as features di-

* Corresponding author.
Email addresses: stahmasebi89@gmail.com (S. Tahmasebipour), babamir@kashanu.ac.ir (S. M. Babamir)
ISSN: 2322-4460 © 2014 JComSec. All rights reserved.

rected at a particular aspect of quality attributes, and (2) we attach simplicity to run time tactics.

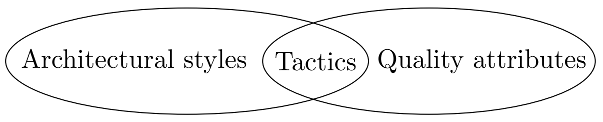

Figure 1. Relation between architectural styles, quality attributes and tactics

An architect uses tactics to an appropriate design using architectural styles. In other words, a set of tactics is implemented in any architectural style. To satisfy the security quality attribute in the Pipe & Filters style, for example, tactic of maintaining data confidentiality is used. This tactic equip a data stream with two filters in its output and input for encryption and decryption respectively.

The Pipe & Filter, Layered, Broker, Blackboard and Client-Server architectural styles and the security, availability and performance tactics are considered in this paper. They are respectively introduced in [2], [9], [10], [11],[12], [13] and [5]. In this study, we deal with the interaction between the tactics and architectural styles mentioned above. Then, we rank the architectural styles using our proposed approach considering the priority of the quality attributes suggested by the software designer.

The paper continues as follows: in Section 2, we discuss related work. In Section 3, we explain the architectural tactics and in Section 4, we explain the interaction between architectural tactics and style. In Sections 5, 6 and 7 we determine the interaction between the architectural security, availability and performance tactics and Pipe & Filter, Layered, Broker, Blackboard and Client-Server architectural styles. In Section 8, we propose ranking the architectural styles using our proposed approach and finally the concluding remarks are given in Section 9.

2 Related Work

In this article, the interaction between tactics and common architectural styles for satisfaction of quality attributes is discussed. Tactics were introduced by [5] to meet quality attributes in architectural styles and applied by some researchers, [14]. For the availability quality attribute, for example, the availability tactic was introduced. This tactic is concerned with software failure and its consequences. A failure occurs when software delivers a service that is not consistent with its specification and such a failure is observable by the software users. The recovery or repair tactic is an important type of availability that prevents faults leading to failures or at least bounds effects of faults and makes repair possible. Availability tactics are divided into three types: a) fault detection, b) fault recovery and c) fault prevention. For fault detection, three tactics were introduced as follows:

- Ping/Echo: One component issues a ping and expects to receive back an echo from the component under scrutiny within a predefined time. If an echo is not returned within the predefined time, the component would be considered as a failed one,
- Heartbeat: In this case, one component emits a heartbeat message periodically and another component listens to the heartbeat. If the heartbeat fails, it is assumed that the originating component has failed.
- Exception: One method for recognizing faults is exception, which is raised when a fault occurs.

Now we deal with contrast between our approach and the related work. Architectural styles and tactics were discussed in [5] but the interaction between them was not studied. The interaction between architectural styles and tactics was addressed in [8] where a few interactions were introduced using six types of changes. To investigate the interaction, they defined concept of *impact magnitude* for each type of change. *Magnitude* of an impact shows how much a style should change to implement a tactic. However, the study of the interaction between the architectural tactics and the set of architectural styles in this paper is different from what studied in [8]; moreover, we address ranking of architectural styles to satisfy quality attributes, which was not presented by [8]. The tactics in embedded systems were proposed by [7]; such systems are not our issue in this paper. Architectural tactics were used for development quality-driven software architecture by [14]; however, the interaction and ranking of styles were not addressed. The interaction between requirements and tactics and architecture patterns was discussed to implement the reliability quality attribute by [15]. However, in this paper the security, availability and performance tactics are considered; furthermore, some other tactics are considered. The safety tactic was applied to the system safety by [16, 17] and some tactics were introduced to improve fault tolerance by [18, 19]. Fault Tolerance Tactics were applied to quantify styles of Shared Repository, Layered, Pipe & Filter, Presentation Abstraction Control, Model View Controller, Broker, Client-Server, and State Transition by [18].

Though others applied quality attributes to quantify software architecture styles, they did not exploit the software architecture tactics. Instances are as follows: In [20], the maintainability quality attribute in architectural styles of Batch Sequential, Shared Data, Abstract Data Type and Implicit Invocation were discussed and a cost based numerical measurement for the quality attribute was obtained by assigning weights to the architecture elements in different architectural styles.

In [21], making suitable decisions were presented for selection of architecture styles of Pipe & Filter, Layered, Blackboard and Abstract Data Type considering the quality attributes of Performance, Flexibility, Reusability and Maintainability. They used the ANP (Analytic Network Process) to support of software architects' decisions. The ANP is the general form of AHP (Analytical Hierarchical Process) addressing complex decisions with interdependences between them.

A method for selection of architecture styles in the Decision Support System (DSS) was presented by [22] where fuzzy inference and styles properties are used to support the selection method.

The AHP-based method for the systematic selection of architecture styles based on characteristics of (a) architectural elements and (b) the target system was presented by [23]. To compare two properties, value of 1 is considered if properties have same level of importance and a maximum value of 9 is done if one property is strongly preferred to another one. The approach in [23] considers 5 architecture styles: Pipe & Filter, Client-Server, Blackboard, Layered and Interacting Process for Data-flow, Hierarchical, Data-centered, Call & Return and Event systems respectively.

In this study: (1) the interaction between the Pipe & Filter, Layered, Broker, Blackboard and Client-server architectural styles and the security, availability and performance tactics is discussed and (2), the common architectural styles are ranked based on the quality attributes by adopting a new proposed approach. Eventually, the best architectural style for every individual quality attribute or a combination of quality attributes is suggested.

3 Architectural tactics

To satisfy each quality attribute, some design and run time tactics have already been presented. The information hiding tactic, for example, is a design time tactic that improves the modifiability quality attribute and the Authenticity tactic is a run time tactic that enhances the security quality attribute. In this study, the run time tactics introduced by [5], are analyzed. Security tactics (Figure 2) are divided into three types: a) resisting attacks, b) detecting attacks and c) recovering from attacks. Figure 3 shows the availability tactics that are divided into three types: a) fault detection, b) fault recovery and c) fault prevention. Performance tactics (Figure 4) are divided into three types: a) resource demand, b) resource management and c) resource arbitration, [5, 24]. The Resisting tactic features including Authentication, Authorization and Confidentiality were explained in [25, 26].

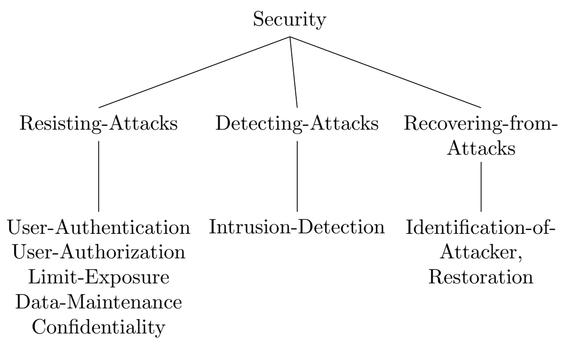

Figure 2. Architectural tactics of security [5]

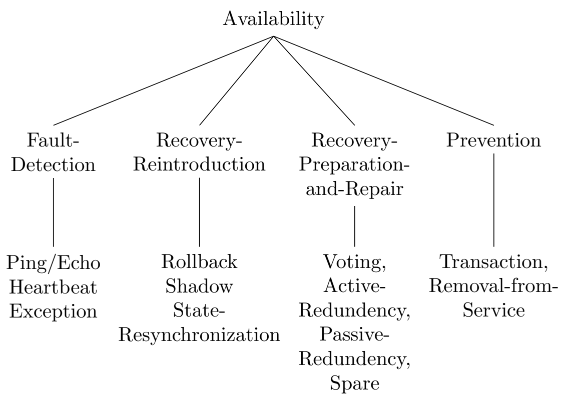

Figure 3. Architectural tactics of availability [5]

4 Interaction Between the Tactics and Software Architectural Styles

As mentioned in Section Introduction, architectural tactics are the intersection points of quality attributes and architectural styles. Indeed, tactics determine how to implement quality attributes in architectural styles. To investigate the interaction between tactics and styles, six types of structure and behaviour changes of the architectural styles were introduced by [8]. When a tactic is to be implemented in an architectural style, the style may require more than one change, as follows: "Implemented in", "Replicates", "Add in the Style", "Add out of the Style", "Modify" and "Delete", which are defined as:

- Implemented in: This tactic is implemented within a component of a style by inserting some actions in the component
- Replicates: A component is duplicated and sequence of actions is copied most likely to different

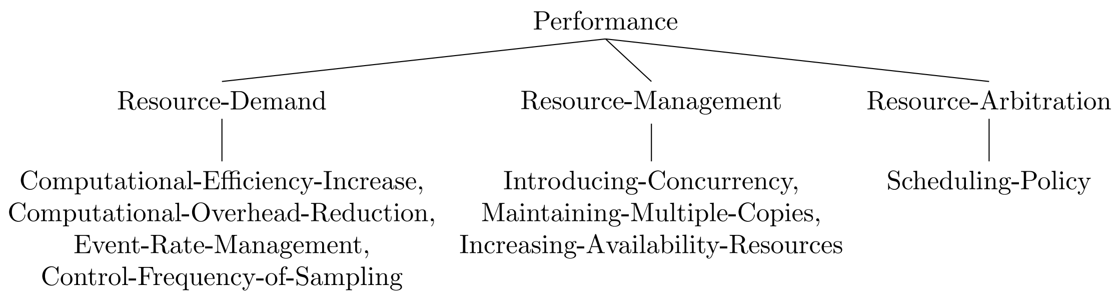
Performance
Resource-Demand
Resource-Management
Resource-Arbitration
Computational-Efficiency-Increase,
Computational-Overhead-Reduction,
Event-Rate-Management,
Control-Frequency-of-Sampling
Introducing-Concurrency,
Maintaining-Multiple-Copies,
Increasing-Availability-Resources
Scheduling-Policy

Figure 4. Architectural tactics of performance [5]

hardware

- Add in the Style: A new instance of a component is added to the software architecture, which adhere to structure of the original architecture style and the style constraints
- Add out of the Style: A new component is added to the architecture but it does not follow the style structure. The behaviour and actions of the added component are different from those of the style
- Modify: The changes of a component structure implies that there are changes in or additions to action sequence of the component
- Delete: A component is removed

The impact magnitude concept was introduced to evaluate the implementation of tactics of architectural styles, [8]. Magnitude of impact shows the change amount of a style to implement a tactic (Table 1). The impact magnitude is a function of the number of required changes, which is determined through five scales:

Table 1. Magnitude of Impact as a function of the number of required changes[8]

Change Type	Change Number	Impact Range
Implemented in	1	+ + to +
Replicate	3 or less	+ + to +
Replicate	more than 3	+ to ∼
Add in the Style	3 or less	+ + to +
Add in the Style	more than 3	+ to ∼
Add out of the Style	3 or less	∼ to -
Add out of the Style	more than 3	- to –
Modify	3 or less	++ to -
Modify	more than 3	∼ to –

- Good Fit (indicated by "+ +"): The style structure should not change to implement the tactic, i.e., the style structure is highly compatible with the requisite structural of the tactic
- Minor Changes (indicated by "+"): The tactic can be implemented by changing the style structure
- Neutral (indicated by "∼"): The style and tactic are basically orthogonal meaning that the tactic is implemented independent of the style and neither receives help from the style nor hinders it
- Significant Changes (indicated by "-"): The needed changes are very significant
- Poor Fit (indicated by "- -"): Compared with previous case, the style structure requires more changes in order to implement the tactic

5 Interaction Between the Security Tactics and Common Architectural Styles

Security tactics are used to meet security quality attributes in the architectural styles. Therefore, interaction between these tactics and architectural styles should be determined. Table 2 shows our findings for interaction between security tactics (see Figure 2) and common architectural styles through magnitudes of −, ++, +, -, ∼ and types of changes (i.e., Implemented in (I), Add in the Style (AI), Add out of the Style (AO) and Modify (M)). As stated in the previous section, in order to determine interaction between the tactics and architectural styles, six types of changes are considered. To implement a tactic in an architectural style, it may require more than one change. The impact magnitude of the interactions are determined according to Table 1.

Implementation and impact of 5 security tactics in the architectural styles are explained as follows:

User Authentication Tactic and Broker Style: As Figure 2 shows, the "user authentication" is a Security tactic. In the Broker style, client is not directly connected to the server and the broker component acts as an interface between the client and server. Thus, the broker component should change so that the tactic "user authenticated" settle in the style. The broker component is naturally an appropriate place for the authentication, inasmuch as this change occurs inside

Table 2. Interaction between security tactics and common architectural styles Legends: **CT**: Change Type, **I**: Impact

Tactic →	Resisting Attacks								Detecting Attacks		Recovering from Attacks			
	Authentication		Authorization		Limit Exposure		Maintenance Data Confidentiality		Intrusion Detection		Attacker Identification		Restoration	
Style ↓	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**
Pipe & Filter	AI,AO	∼	AO,M	- -	I	++	AI	+	AO,M	- -	AO,M	- -	AO,M	- -
Layered	AI	++	M	- -	I	++	AI,M	∼	AO,M	- -	AO,M	- -	I	++
Broker	I	++	M	-	M	- -	AO,M	- -	M	+	AO,M	- -	I	++
Blackboard	AO,M	- -	I	+ +	AO	-	AO,M	- -	AO,M	- -	AO,M	- -	M	- -
Client-Server	AI	+	AI,M	+	M	- -	AI,M	-	AO	- -	AO,M	- -	I	++

the broker component and there is a connection between the client and broker. Consequently, this tactic lies in the "Implemented in" category. In this case, one component changes and its impact magnitude is "++" (see Table 1).

User Authorization Tactic in and Blackboard Style: The blackboard style includes a control component for authorizing users; so, type of this change is "Implemented in", inasmuch as the control component controls the blackboard routinely. Since only this component changes, the impact magnitude is "++" (see Table 1).

Maintaining data confidentiality Tactic and Layer Style: As Figure 2 shows, "maintaining data confidentiality" is a Security tactic. If a layered software should operate within a secure environment and all the software layers belong to a process, a layer would be responsible to encrypt the software output. In this case, the type of change is "Add the in Style", inasmuch as it follows the style structure. Since a layer is added, regarding the Table 1, its impact is "++". If the layered software operates within an insecure environment and each layer is a separate process, the layers should change so that the output of each layer is encrypted; accordingly, each layer input data should be decrypted before processing (starting from the bottom layer). Such changes are called the "Modify" type because layers should change for the encryption and decryption purposes. Since all layers should be changed and the number of layers is usually more than 3, its impact magnitude is "–" (see Table 1). Thus, the impact magnitude of this tactic depends on the layers implementation and it is positive or negative. Consequently, the final impact magnitude for this tactic is "∼".

Intrusion detection Tactic and Client-Server Style: As Figure 2 shows, the "Intrusion detection" is a Security tactic. In this tactic, opening software ports should be supervised; this is implemented as an independent process. A monitor component is added to monitor the software opened ports. The type of this change is "Add out of the Style", inasmuch as the added monitor component (along with its connections to the style components) does not follow the style structure. Since the number of changes is more than 3, the impact magnitude of this tactic is "–" (see Table 1).

Restoration Tactic and Pipe & Filter Style: As Figure 2 shows, "Restoration" is a Security tactic. In this tactic, the compatible state should be saved and the incompatible state should be restored to the compatible state. However, the "Pipe & Filter" style has no state and it acts on a data stream. Consequently, a component should be added for monitoring the style operation in order to save software states. Therefore, types of changes of this tactic are as follows: because of using the monitor component, type of change is the "Add out of the Style", which it does not follow the style structure. For a filter component, type of change is "Modify", inasmuch as filters should be modified to send their states to the monitor component. Thus, we can conclude that the number of changes is more than 3 and the impact magnitude is "- -" (see Table 1).

6 Interaction Between Availability Tactics and Common Architectural Styles

Availability tactics are used to meet the availability quality attribute of the architectural styles. Therefore, the interaction between availability tactics and architectural styles should be determined. Tables 3 and 4 show our findings about the availability tactics. The tables show the interaction between availability tactics and architectural styles of Pipe & Filter, Layered, Blackboard, Client-Server and Broker. As stated in Section 5, the interaction between tactics and architectural styles leads to six types of changes in the structure and behavior of architectural styles. For the tactic implementation in an architectural style, it may require more than one change.

The impact magnitude of the interaction is determined according to Table 1. Types of changes are: Implemented in (I), Replicates (R), Add in the Style (AI), Add out of the Style (AO), Modify (M) and mag-

Table 3. Interaction between availability tactics and common architectural styles Legends: **CT**: Change Type, **I**: Impact

Tactic →	Fault Detection						Recovering/Reintroduction					
	Ping/Echo		Heartbeat		Exception		Checkpoint Rollback		Shadow		Stat Resynchronization	
Style ↓	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**
Pipe & Filter	AO,M	- -	AO,M	- -	AO,M	- -	AO,M	- -	AI	+	AO,M	- -
Layered	AI,M	+	AI,M	+	I	++	I	++	R,AI	+	M,AI	+
Broker	I	++	M	+	M	+	I	++	I	++	I	++
Blackboard	M	∼	M	∼	M	- -	M	- -	AO,R	-	M	+
Client Server	AI,M	+	AI,M	+	M	+	I	++	AI,R	+	AI,M	+

Table 4. Continued from Table 3

Tactic →	Recovery,Preparation and Repair						Prevention					
	Voting		Active Redundancy		Passive Redundancy		Spare		Transaction		Removal from Service	
Style ↓	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**
Pipe & Filter	AI	+	R	+	R,M	-	AI	+	M	∼	AO	- -
Layered	AI,M	+	R,AI	+	M,R	-	M,R	∼	I	++	M,AI	∼
Broker	I	++	I	++	M	+	I	++	I	++	I	++
Blackboard	AI,M	+	R,M	+	M,R	∼	M,R	∼	M	- -	AO	- -
Client Server	AI	+	AI,R	+	AI,R,M	+	AI,I	+	I	++	AO	∼

nitude of impact are: −, ++, +, -, ∼ (see Tables 3 and 4). Implementation and impact of 5 availability tactics in the architectural styles are as follows:

Ping/ Echo Tactic and Layer Style: As Figure 3 shows, this tactic is an Availability one. In this tactic, an additional component is required to send the ping massage. If all layers belong to one process, the top layer responds to the sender layer by the ping message, which called the echo message. If each layer is a separate process, each layer sends the ping message to its adjacent lower layer before echoing the response to its upper layer. In fact, ping and echo messages are exchanged hierarchically. Thus, this change lies in the "Add in the Style" type because it follows the style structure. On the other hand, to respond to the ping message via the echo message, layers should change; so, this change lies in the "Modify" category. Regarding Table 1, the final impact magnitude of this tactic is "+", inasmuch as the number of changes is more than one. The first change is adding a layer for sending the ping message with the impact magnitude "++" and the second one is the change of layers to respond via the echo message with the impact magnitude "-".

Checkpoint and roll back Tactic and Blackboard Style: As Figure 3 shows, this tactic is an Availability tactic. Since determination of checkpoints: (1) is awkward for rolling back and (2) it contradicts the blackboard idea, an alternative method should be used to log all inputs. Nevertheless, this method is awkward and needs large storage. Therefore, the blackboard component requires significant changes. In this case, the type of change is "Modify" and the number of changes is more than 3. Thus the impact magnitude is "–" (see Table 1).

Shadow Tactic and Broker Style: As Figure 3 shows, this tactic is an Availability tactic. In the Broker style, the broker component is a suitable component for monitoring servers. It saves states of servers and marks them as shadow until they return to their normal operations. Since the broker component typically controls servers operations, the type of change is "Implemented in" and its impact magnitude is "++" (see Table 1).

Voting Tactic and Pipe & Filter Style : As Figure 3 shows, this tactic is an Availability tactic. In the Pipe & Filter style, Filters are considered as voter and furthermore a specific filter is considered as voting mechanism. Since input of a filter should be distributed among the voters, a pipe with one input and some outputs should be used. Since the style structure remains unchanged and some additional filters are added, this change lies in the "Add in the Style" type. As the number of added components is more than 3 (a couple of voter filters and a voting filter), the impact magnitude is "+" (see Table 1).

Active Redundancy Tactic and Client-Server Style: As Figure 3 shows, this tactic is an Availability tactic. In this tactic, servers are replicated and a central component is required for sending a request to each server and for concurrent distribution of the request among redundant servers. Response of the main and redundant servers are sent to the client and central component respectively. If a fault occurs, the result is sent to the client via the central component. The central component plays the client role to the server. Since these changes lead to the style structure, types of the changes are: "Add in the Style" due to incorporation of the central component to the style and "Replicates" due to replication of the servers. Since the number of changes is 2 at least, the impact magnitude is "+" (see Table 1).

7 Interaction Between Performance Tactics and Common Architectural Styles

Since the performance tactics are used to meet the performance quality attributes in the architectural styles, the interaction between performance tactics and architectural styles should be determined. Table 5 shows our findings about the performance tactics where the interaction between performance tactics and architectural styles of Pipe & Filter, Layered, Blackboard, Client-Server and Broker is determined. As stated in Section 5, in order to describe the interaction between tactics and architectural styles, six types of changes in the structure and behaviour of architectural styles were introduced. For the implementation of a tactic in an architectural style, it may require more than one change and the impact magnitude of the interaction is determined according to Table 1. Types of changes include Implemented in (I), Replicates (R), Add in the Style (AI), Add out of the Style (AO) , Modify (M). Table 5 shows the magnitude of impacts (–,++,+,-,∼) for performance tactics (see Figure 3).

Implementation and impact of 5 performance tactics in the architectural styles are explained as follows:

Increasing computational efficiency Tactic and Layer Style: As Figure 4 shows, this tactic is a Performance tactic. In the Layer style, each layer performs a separate task; accordingly, the embedded algorithms in layers should be optimized to increase the computational efficiency. To this end, all layers should change. Thus, the type of this change is "Modify". Since some layers change, its impact magnitude is minus "-" (see Table 1).

Computational overhead reduction Tactic and Blackboard Style: As Figure 4 shows, this tactic is a Performance tactic. To implement this tactic, a monitor component is required to monitor knowledge sources. If a knowledge source has no request for processing, it would be removed from the service. Therefore, the need to processing requests are reduced leading to decrease computational overhead. Because the monitor component does not follow the style structure, the type of change is "Add out of the Style". The monitor component along with its connections to knowledge sources are added and the knowledge sources should be changed; so, the type of this change is "Modify". Thus the number of changes is more than 3 and its impact magnitude is "- -" (see Table 1).

Introducing Concurrency Tactic and Pipe & Filter Style: As Figure 4 shows, this tactic is a Performance tactic. In this style, filters should act in parallel when requests arrive. Since filters normally process requests in parallel, the type of change is "Implemented in" and regarding the Table 1, the impact magnitude is "+ +".

Maintaining multiple copies Tactic and Client-Server Style: As Figure 4 shows, this tactic is a Performance tactic. To implement this tactic in this style, the server is copied to the client side so that it becomes a local server for the client. The main server updates are sent to the copied server and so the main server changes; accordingly, the change type is "Modify". Indeed, during sending updates, the main server plays role of a server and the copied server plays the client role. Therefore, the style structure remains unchanged. This change type is "Replicates" because the server was copied. Since some copies were added to the style, the impact magnitude is "+" (see Table 1).

Scheduling Policy Tactic and Broker Style: As Figure 4 shows, this tactic is a Performance tactic. In this tactic, requests are received by the broker component at first. Therefore, the component should change to enable itself to apply the scheduling policy for the requests. On the other hand, servers should change to accept the scheduling policy. Since both the broker component and the server change, this change type is "Modify". Consequently, the number of changes is less than 3, and accordingly, the impact magnitude is "+" (see Table 1)

Table 5. Interaction between performance tactics and common architectural styles Legends: **CT**: Change Type, **I**: Impact

Tactic →	Resource Demand								Resource Management				Resource Arbitration			
	Computational Efficiency Increase		Computational Overhead Reduce		Event Rate Management		Sampling Control Frequency		Concurrency Introduction		Multiple Copies Maintenance		Resources Available Increase		Policy Scheduling	
Style ↓	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**	**CT**	**I**
Pipe & Filter	M	-	M,AO	- -	M,AO	- -	AI	+	I	++	AO,R	- -	M	∼	M	- -
Layered	M	-	AO	-	M,AI	∼	AI	+	R,AI	+	M	-	M	∼	M	- -
Broker	M	+	M,AO	- -	AO,M	-	M	+	R,M	∼	AO,R	∼	M	∼	M	+
Blackboard	M	+	AO,M	- -	AO	- -	M	+	I	++	AO,M,R	- -	M	∼	M	+
Client Server	M	+	M,AO	- -	M,AO	-	AI	+	R	+	R,M	+	M	∼	AO,M	- -

8 Common Architectural Styles Ranking to Satisfy the Quality Attributes

We calculate the average of impact magnitude for each style using Tables 2, 3, 4 and 5 and by assigning a number to each impact magnitude scale (which was introduced in Section 5). In these tables, we use 5 point scale adopted from [23]where the notations of "++", "+", "–", "-" and "∼" are replaced by numbers +2, +1, -2, -1 and zero respectively, [23]. The row average for the mentioned tables is given in Table 6.

Table 6. Average of impact magnitude of tactics in common architectural styles

	Security	Availability	Performance
Pipe&Filter	-0.714	-0.75	-0.75
Layered	0	0.917	-0.375
Broker	-0.286	0.175	0
Blackboard	-1.286	-0.5	-0.125
Client-Server	-0.428	1.083	-0.125

In an architectural style, being more positive signs in a tactic impact magnitude indicates that the style structure has a better consistency to implement the tactic. According to the Table 6, the consistency of common architectural styles in implementation of (a) security, (b) availability and (c) performance tactics are:

a) Layer, Broker, Clientserver, Pipe & Filter and Blackboard
b) Broker, ClientServer, Layered, Blackboard and Pipe & Filter
c) Broker, ClientServer, Blackboard, Layered and Pipe & Filter

We explained the interaction of the security, availability and performance tactics with common architectural styles in Sections 6 to 8. Then, we calculated the impact average of all mentioned tactics where the impact average indicates the style consistency in implementation of the tactics. The tactics determine how to implement quality attributes in the architectural styles. To rank an architectural style based on the intended quality attributes, we define Table 6 as the following Equations.
In the next step, we define priority of security, availability and performance as a matrix, say, $q_{3\times 1}$ based on the software developers' demand. A weight at [0,1] interval is assigned to each quality attribute in range of 0% to 100% by the software developers. The first three entries in matrix q indicates security, availability and performance quality attributes, respectively. In the next step, we suggest Equation 1 to rank the architectural styles based on the desired quality attributes.

$$R = ST \times q \tag{1}$$

Each one of the entry values in matrix R indicate an architectural style compatibility. Indeed, each matrix entry in R determines rank of an architectural style in satisfaction of an intended quality attribute where the more positive entries indicates the higher rank. The evaluation is presented with a case study where requests of mail order companies are collected.

8.1 Case study: Collective Ordering System

Mail order companies prefer to work with collective orderers who accumulate and submit customers orders to a mail order company, [27]. The company delivers goods in a shipment to the collective orderer for distribution. The system features insist of a fact that: The collective orderer knows the formalities and processes for procedures such as reshipment, complaint and deferred payment. The personal contact to customers has positive effect on the business volume. The mail order company can delegate communication activities with customers to the collective orderer. These procedures are affected by frequent changes. Therefore, such a system is chosen as a case study. To design the collective ordering software, the following states are considered:

a. Given that the software developer has attached important (weight) of 65% and 35% to the system security and performance respectively; Eq. 2 is obtained.

$$q_a = \begin{bmatrix} 0.65 \\ 0 \\ 0.35 \end{bmatrix}. \quad (2)$$

Using Equation 1 and replacing q by Equation 2, the styles rank is determined by Equation 3.

$$R_a = \begin{bmatrix} -0.726 \\ -0.1312 \\ -0.1859 \\ -0.8796 \\ -0.3219 \end{bmatrix}. \quad (3)$$

Equation 3 shows that the most positive number is assigned to the layer style. Consequently, to design this software the layer style is the best choice.

b. If weights (important) of security and availability are suggested as 60% and 40% by the software developer respectively, then the following Eq. 4 is obtained:

$$q_b = \begin{bmatrix} 0.6 \\ 0.4 \\ 0 \end{bmatrix}. \quad (4)$$

Using Equation 1 and replacing q by Equation 4, the rank of styles is determined by Equation 5.

$$R_b = \begin{bmatrix} -0.728 \\ +0.376 \\ +0.528 \\ -0.972 \\ +0.176 \end{bmatrix}. \quad (5)$$

Equation 5 shows that the broker style is the most positive one; therefore, it is suggested to the software developer.

c. If weights, 70% and 30% are suggested for availability and performance respectively by the system developer, then Eq. 6 is obtained:

$$q_c = \begin{bmatrix} 0 \\ 0.7 \\ 0.3 \end{bmatrix}. \quad (6)$$

Using Equation 1 and replacing q by Equation 6, styles rank is determined by Equation 7.

$$R_c = \begin{bmatrix} -0.75 \\ +0.5249 \\ +1.225 \\ -0.3875 \\ +0.7206 \end{bmatrix}. \quad (7)$$

That the broker style is the most positive one is shown here; therefore, it is suggested to the software developer.

d. If weights of 70%, 10% and 20% are suggested for security, availability and performance by the software developer respectively, then q in Eq. 1 is replaced by Eq. 8.

$$q_d = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.2 \end{bmatrix}. \quad (8)$$

Using Equation 1 and replacing q by Equation 8, the styles rank is determined by Equation 9.

$$R_d = \begin{bmatrix} -0.725 \\ +0.017 \\ -0.025 \\ -0.975 \\ -0.216 \end{bmatrix}. \quad (9)$$

That the Layered style is the most positive one is shown here; therefore, it is suggested to the software developer.

9 Conclusion

Architectural tactics are intersection point of quality attributes and architectural styles. Indeed, architectural tactics determine how to implement quality attributes in architectural styles. In this article, the interactions between “security”, “availability” and “performance” tactics and common architectural styles were determined. According to the average of values that were obtained through the interaction between architectural tactics and architectural styles, the compatibility of common architectural styles in implementation of, (a) “security”, (b) “availability” and (c) “performance” tactics are as follows, respectively:

a) Styles Layered, Broker, ClientServer, Pipe & Filter and Blackboard
b) Styles Broker, ClientServer, Layered, Blackboard and Pipe & Filter
c) Styles Broker, ClientServer, Blackboard, Layered and Pipe & Filter

Moreover, a new approach was proposed to rank common architectural styles based on the intended quality attributes. Using this approach, software developers can determine the best architectural style for every individual quality attribute or their combinations. In this approach, the following steps were taken:

- Determining the interactions between mentioned tactics and common architectural styles,
- Calculating and defining the average impact of tactics as a matrix,
- Defining the quality attributes priority of security, availability and performance as a matrix based on the software developers demands,
- Acquiring the best architectural style via definition of a new equation.

Acknowledgements

Authors like to thank University of Kashan for supporting this study by Grant No. 92/6188.

References

[1] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures: Views and Beyond*. SEI series in software engineering. Addison-Wesley, Boston, MA, 2003. ISBN 9780201703726.

[2] Nick Rozanski and Eoin Woods. *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, USA, Massachusetts, 2005. ISBN 0321112296.

[3] Mohamad Kassab. *Non-Functional Requirements: Modeling and Assessment*. VDM Verlag, Germany, 2009. ISBN 3639206177, 9783639206173.

[4] K. Saleh and A. Al-Zarouni. Capturing non-functional software requirements using the user requirements notation. In *The International Research Conference on Innovation in Information Technology (IIT'04)*, pages 222–230, 2004.

[5] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, USA, Massachusetts, 3rd edition, 2012. ISBN 0321815734, 9780321815736.

[6] S.H. Al-Daajeh, R.E. Al-Qutaish, and F. Al-Qirem. A tactic-based framework to evaluate the relationships between the software product quality attributes. *International Journal of Software Engineering*, 5(1):5–26, 2012.

[7] S.H. Al-Daajeh, R.E. Al-Qutaish, and F. Al-Qirem. Engineering dependability to embedded systems software via tactics. *International Journal of Software Engineering and Its Applications*, 5(4):45–62, 2010.

[8] Neil B. Harrison and Paris Avgeriou. How do architecture patterns and tactics interact? a model and annotation. *Journal of Systems and Software*, 83(10):1735–1758, 2010.

[9] Paris Avgeriou and Uwe Zdun. Architectural patterns revisited - a pattern language. In *EuroPLoP*, pages 431–470, 2005.

[10] Valérie Issarny and Jean-Pierre Benâtre. Architecture-based exception handling. In *HICSS*, 2001.

[11] George Fairbanks and D. Garlan. *Just Enough Software Architecture: A Risk-Driven Approach*. Marshall & Brainerd, USA, 2010. ISBN 9780984618101.

[12] M. Völter, M. Kircher, and U. Zdun. *Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware*. Wiley Software Patterns Series. Wiley, England, 2013. ISBN 9781118725856.

[13] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill series in computer science. McGraw-Hill Higher Education, USA, 7th edition, 2010. ISBN 9780071267823.

[14] Suntae Kim, Dae-Kyoo Kim, Lunjin Lu, and Sooyong Park. Quality-driven architecture development using architectural tactics. *Journal of Systems and Software*, 82(8):1211 – 1231, 2009. ISSN 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2009.03.102.

[15] Neil B. Harrison and Paris Avgeriou. Implementing reliability: The interaction of requirements, tactics and architecture patterns. In *WADS*, pages 97–122, 2009.

[16] Weihang Wu and T. Kelly. Safety tactics for software architecture design. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 368–375 vol.1, 2004. doi: 10.1109/CMPSAC.2004.1342860.

[17] Weihang Wu and Tim Kelly. Failure modelling in software architecture design for safety. *SIGSOFT Softw. Eng. Notes*, 30(4):1–7, May 2005. ISSN 0163-5948. doi: 10.1145/1082983.1083222.

[18] Neil B. Harrison and Paris Avgeriou. Incorporating fault tolerance tactics in software architecture patterns. In *SERENE*, pages 9–18, 2008.

[19] N. Harrison, P. Avgeriou, and Uwe Zdun. On the impact of fault tolerance tactics on architecture patterns. In *2nd International Workshop on Software Engineering for Resilient Systems (SERENE 2010)*, New York, USA, April 2010. ACM.

[20] H. Hashemian and F. Shams. A model for investigating maintainability quality attribute in soft-

ware architecture styles. In *The 4th Malaysian Software Engineering Conference*, 2008.

[21] M. K. Delhi Babu, P. Govinda Rajulu, A. Ramamohana Reddy, and A. N. Aruna Kumari. Selection of architecture styles using analytic network process for the optimization of software architecture. *CoRR*, abs/1005.4271, 2010.

[22] Shahrouz Moaven, Jafar Habibi, Hamed Ahmadi, and Ali Kamandi. A decision support system for software architecture-style selection. In *SERA*, pages 213–220, 2008.

[23] Matthias Galster, Armin Eberlein, and Mahmood Moussavi. Systematic selection of software architecture styles. *IET Software*, 4(5):349–360, 2010.

[24] Roger Champagne and Sebastien Gagné. Towards automation of performance architectural tactics application. In *WICSA*, pages 157–160, 2011.

[25] F.S. Babamir and Z. Eslami. Data security in unattended wireless sensor networks through aggregate signcryption. *TIIS*, 6(11):2940–2955, 2012.

[26] F.S. Babamir and A. Norouzi. Achieving key privacy and invisibility for unattended wireless sensor networks in healthcare. *The Computer*, 2013. Oxford University, doi: 10.1093/comjnl/bxt046, published online (May 8, 2013).

[27] Stephan Bode and Matthias Riebisch. Impact evaluation for quality-oriented architectural decisions regarding evolvability. In *ECSA*, pages 182–197, 2010.

Safieh Tahmasebipour received a Bachelor of Science degree in software engineering from the University of Qom, Qom, Iran, 2010 with the thesis of “Simulation of nanoscale electrical discharge using MATLAB”. She received a Master of Science degree in Software Engineering from the University of Kashan, Kashan, Iran, 2012 with the thesis of “Quality evaluation of software architecture styles using architectural tactics”. She has already authored 6 international and internal conference papers.

Seyed Morteza Babamir received BS degree in Software Engineering from Ferdowsi University of Meshhad and MS and PhD degrees in Software Engineering from Tarbiat Modares University in 2002 and 2007 respectively. He was a researcher at Iran Aircraft Industries, Tehran, Iran, from 1987 to 1993, head of Computer Center in University of Kashan, Kashan, Iran, from 1997 to 1999 and head of Computer Engineering Department in University of Kashan from 2002 to 2005. He is associate professor of Computer Engineering Department in University of Kashan. He authored one book in Software Testing, 4 book chapters, 20 journal papers and more than 50 international and internal conference papers (`http://ce.kashanu.ac.ir/babamir/Publication.htm`). He is managing director of Soft Computing Journal published by supporting University of Kashan, Kashan, Iran.