Assessing Architectural Patterns Trade-offs using Moment-based Pattern Taxonomies

Cristian Orellana, Mónica M. Villegas, Hernán Astudillo
Toeska Research Group
Universidad Técnica Federico Santa María
Valparaíso, Chile
Email: cristian.orellanae@usm.cl, monica.villegas@usm.cl, hernan@inf.utfsm.cl

2019 XLV Latin American Computing Conference (CLEI) 978-1-7281-5574-6/20/$31.00 ©2020 IEEE 10.1109/CLEI47609.2019.235086

***Abstract*—Large software systems are designed to satisfy or accommodate many requirements; architectural patterns are a well-known technique to reuse design knowledge. However, requested quality attributes (QA) may be inconsistent at times; e.g., high security typically hampers performance and scalability. Thus, a key concern of systems architects is understanding trade-offs among alternative solutions; e.g., a pattern may favor performance at the expense of scalability or security, another may privilege scalability, and yet another may push security. This article argues that the usual organization of individual patterns in topic-related pattern languages is not too helpful to identify trade-offs, and proposes to borrow a taxonomic principle of architectural tactics, organizing the patterns for each QA into "moments". This enables architects to use simple trade-off highlighting techniques to understand trade-offs in complex systems. The approach was used in the systematic design of a SCADA-to-ERP secure bridge, where moment-oriented pattern taxonomies for availability, confidentiality, and performance were used. This approach offers the promise of enabling the trade-off-enabled, pattern-driven design of large systems by supporting the systematic exploration of trade-offs among patterns for specific QA's.**

***Index Terms*—Design trade-offs, Architectural patterns, Software architecture tactics**

I. Introduction

Designing the structure of a system that illustrates the correspondence between the requirements and the built software is one of the main challenges of software architecture. A set of predefined structures as solutions to recurring problems has been described and characterized through patterns. [26]. Design patterns [25] are "micro-architectures" while software architectures are more coarse-grained designs.

A pattern is described as a solution to a class of problems in a general context [15]. Architectural patterns have been used over time as a software design technique that allows systems to be structured as blocks, exhibiting more structured systems and hence, more maintainable systems through time.

However, the selection and comparison of suitable patterns to design a system is not a trivial process, since there are several problems to be solved: completeness of the descriptions, the scope of the patterns catalog, use of multiple catalogs, among others. Also, the selection of appropriate patterns for a specific context is usually a problem for novice software engineers [27].

However, some situations allow the architect to discern about the relevance of one pattern over another according to how they impact Quality Attributes. A Quality Attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders [15].

Finding that there are several architectural patterns available, choosing the right ones based on the requirements and characteristics of the systems we need to design, becomes a hard task. Also, the comparison and selection of these patterns turn into a complicated situation because there might be a lack of criteria for the right selection [28].

For this problem, we found it useful to design and propose a method which aims to reduce the space of solutions for our architectural problems. This method allows us to have a more systematic selection of patterns based on mapping patterns around architectural tactics categories and then identifying the trade-offs between them to enable the right decision making regarding the patterns selection.

We introduce the concept of "moment" as a new criterion to classify architectural patterns. A "moment" corresponds to an existing category of a taxonomy of architectural tactics for a specific QA but applied to classify architectural patterns. An architectural tactic is a design decision that influences the achievement of a quality attribute response [15]. We consider that these categories correspond to instants of time that take place during the execution of the software, and depending on that moment, the system must react to a specific stimulus or avoid future scenarios that could compromise a QA. We argue that this classification allows us to discern more clearly in the selection of patterns and identify associated trade-offs between patterns that are classified in the same "moment".

Our proposal describes a systematic approach that allows (1) to classify patterns according to known taxonomies (from architectural tactics), (2) analyze the trade-offs of each pattern, and (3) finally derive a set of appropriate patterns for the solution that is required, significantly reducing the space for alternative solutions.

The remainder of this document is structured as follows: Section II explores the efforts and related work by analyzing state of the art; Section III identifies trade-offs associated with pattern selection; Section IV illustrates the proposal through an illustrative case; Section V analyzes the applicability of

our proposal and the scope of it; and finally, Section VI summarizes and concludes.

II. Related Work

In software architecture, we can consider patterns as first-class citizens that constitute the building blocks of the software to be built. Using a set of appropriate patterns, a design can be derived according to the system requirements. But this has important challenges: (1) pattern selection [4], and (2) analysis of solution alternatives considering trade-offs for each specific case.

A. Pattern Selection

Pattern selection is a task whose difficulty is closely related to the experience of software engineers [28]. Some articles describe the approaches and proposals that try to solve the intrinsic challenges of the tasks associated with the selection of patterns [4].

Some papers describe approaches to select patterns that make use of several techniques such as Case-Based Reasoning (CBR) and Formal Concept Analysis (FCA), Information Retrieval (IR) Techniques, Implicit Culture (IC) [27]. However, this method requires adapting to each of the steps described before you can support with pattern selection. It also lacks flexibility and does not consider associated trade-offs.

In the literature, it is possible to find articles that describe automated frameworks to support pattern selection [29]. However, the formal specification of the problem and the large sample size that is required for the learning process that is carried out through the techniques implemented in this framework are issues that discourage its use.

Other papers explore the use of patterns as a design aid, but without conclusive results: an experimental study focused on safety patterns did not allow to conclude that the use of patterns is helpful for designers [2].

B. Trade-offs and Patterns

When identifying alternative solutions, an architect must discern if an option affects other components of the system and if their choice impacts positively or negatively on any QA [22]. These design decisions are closely related to trade-offs and are a subject of study of software architecture.

We can define "trade-off" as *"an architectural decision that affects two or more system properties, making at least one property better and at least one property worse"* [22].

Zhu [1] uses AHP (Analytical Hierarchy Process)[16] to perform sensitivity analysis by identifying trade-offs, but without specifying how decisions affect a QA using a quantitative approach. That is, it is not specified if an election impacts positively or negatively on intermediate decisions.

Beck [3] argues that it is possible to derive architecture from patterns, but omits the exploration of different solution alternatives. Other authors [5] have defined some metrics to evaluate solution alternatives, that allow deriving design patterns for specific solutions; however, the technique is illustrated only theoretically, and there may be a gap when applied in real cases.

Several techniques have been proposed to compare alternative solutions. NFR Framework [6] is a softgoal-based approach to explore solution alternatives that address a given QA, using softgoals. This technique allows documenting the rationale underlying a design to satisfy a given QA, and its use has been illustrated in several works to identify trade-offs in a defined solutions space [7].

Other approaches highlight the importance of architectural decisions and how they can be addressed through patterns [8]. These have constituted an area called *Architectural Knowledge* whose focus is to identify, describe, justify and support design decisions by architects [9]

Attribute-Driven Design (ADD) is a method for designing the software architecture of a system or collection of systems based on an explicit articulation of the quality attribute goals for the system(s) [21]. ADD+ is an extension to the ADD method by incorporating the role of system stakeholders in making architecture trade-off decisions [20].

None of these techniques allows exploring in-depth alternative pattern-based solutions, perhaps because a good characterization of a solution's trade-offs requires considering the problem context. Only the classification of the patterns according to addressed QA's is not enough. System designers who want to use patterns, need support to compare and select alternative patterns that provide capabilities for specific moments in the system's execution, e.g., detect, react, mitigate or recover from an attack.

III. Trade-off-driven Patterns Selection

We propose a systematic method that derives configurations of patterns according to the context of the problem, based on QA's that the system must exhibit. The key piece is a catalog of patterns for each QA, filtered regarding project constraints. Patterns are organized according to "moments" in the system's execution, which we have found to be more precise.

Designers can use this trade-off information to compare and select specific patterns to be deployed for each moment.

Our method proposes to address the following steps:

1) Review Business Problem and Architecture Problem (Quality Attributes, domain-specific restrictions, etc.)
2) Reduce options systematically
3) Map patterns into a QA-based taxonomy
4) Identify trade-offs

The following subsections describe each step.

A. Review Business Problem and Architecture Problem

The business problem corresponds to an undesired situation that must be improved or addressed by incorporating a technological solution. However, the effort in this step is focused on characterizing this situation, identifying domain elements that are critical for specific processes and that have a close relationship with the unfavorable state that deserves to be improved. In this step, we try to characterize the problem and not its solution; then, it is not appropriate to propose

preliminary technological solutions that could eventually solve the problem.

The architecture problem derives from transforming the business problem into a technological problem. At this stage, we are interested in being able to identify QA's and domain-specific constraints that are critical for the design of the system, without the need to address other aspects of the architecture problem.

The output of this step corresponds to a set of QA's and project constraints that the system must exhibit and patterns must support that. Well-known QA's are *Confidentiality*, *Modifiability*, and *Scalability*.

B. Systematic reduction of options

In this step, it is required to have a catalog of patterns for a specific QA. Since a system must generally address multiple QA's, there is a significant challenge that consists of finding a catalog of patterns that address multiple QA's, and whose description allows a comparison between the patterns that this catalog has.

Another option to support the selection of the pattern catalog is to combine descriptions of patterns from different catalogs; this is only possible if these descriptions are structurally similar and have a similar level of abstraction and completeness.

It is possible to find literature on different catalogs of architectural patterns that allow addressing certain QA's. Fowler et al. [10] describe a set of architectural patterns that address recurrent problems in the field of software architecture. Fernandez et al. [11] propose a list of patterns that allow the design of security-oriented systems.

Many of the approaches that allow guiding the selection of patterns tend to be specific to the context, and QA's that need to be addressed. In this way, some articles describe guidelines for the selection of security patterns [14], some cases illustrate industrial contexts with SCADA (Supervisory Control And Data Acquisition) ecosystems [12], while it is also possible to find automated methods for the selection of design patterns [13]. However, the problems that address design patterns differ from those addressed by architectural patterns, which seek to cover QA's in systems that must be built.

An architectural pattern delineates the element types and their forms of interaction used in solving the problem [15]. Therefore, this type of techniques may not be relevant to support the selection of architectural patterns, delegating the responsibility to the software architect.

First, we propose to reduce the decision space through the exploration of the relationships that exist between the patterns belonging to the catalog. The descriptions of the patterns will be used to filter and compare if the patterns described are substitutes, complements, or are beyond the scope of the project requirements. After the above, only one subset of the catalog patterns will be used.

We argue that this subset of derived patterns after applying filtering rules may be non-representative as an eventual possible solution alternative for system design. This is due to the fact that a large part of the design must demonstrate that the system addresses and satisfies the desired QA's, but the system must be designed to respond in runtime to events that involve preventing, reacting, stopping and managing certain stimulus or threats that could compromise a QA that the system must satisfy. Our approach holds that the above can be addressed as long as the system can react to events that happen in moments of time (what we call "moments") and that involves deciding what to do before, during and after they occur.

C. Map patterns into a QA-based taxonomy

The "decisions to be made" approach to addressing a QA is well known in the architectural tactics research [15]. Taxonomies of architectural tactics associate a set of decisions that will be taken at certain times to address a QA. We consider it useful to apply this concept that originally comes from architectural tactics to the field of architectural patterns, to provide a systematic approach that allows us to classify, evaluate and compare several pattern configurations for a specific QA.

We will illustrate this concept with an example that uses a taxonomy of patterns for a system that must exhibit *Integrity* and *Usability*.

Integrity is usually considered a security dimension, and its purpose is to guarantee that stored data and communications cannot be altered by those who do not have authorization [23].

Usability is related to the user experience and the system capabilities to facilitate such interaction.

For illustrative purposes, we will refer to the architectural patterns generically, to indicate that these patterns belong to a specific taxonomy and describe their respective classification according to moments. We will assume that these patterns come from a previously selected catalog.

To design a system that exhibits such QA's, it is necessary to keep in mind that there is a stimulus for each QA that is approached based on moments.

1) Integrity: This QA represents all the design decisions that were considered to assure that a system performs its intended function in an unimpaired manner, free from deliberate or inadvertent unauthorized manipulation of the system [23].

Considering *Integrity* as a *Security* dimension [23], the moments that we will use to classify Integrity Patterns correspond to the categories of the taxonomy of Security Tactics described in [24].

The stimulus to be considered is an "attack" and can be managed through the following moments:

- Detect attack: The system must be able to recognize an attack that potentially compromises the integrity of a system.
- Stop or mitigate attack: The system must be able to stop or mitigate an attack that compromises the integrity of a system
- React to attacks: The system must be able to make decisions when an attack occurs at runtime.
- Recover from attacks: The system must exhibit resilience before an attack that compromises the integrity of a system

Using this approach, it is possible to use a taxonomy that classifies the patterns according to the moments described above. Figure 1 illustrates the taxonomy associated with Integrity Patterns. Each pattern is associated with one or more specific categories (”moments”).

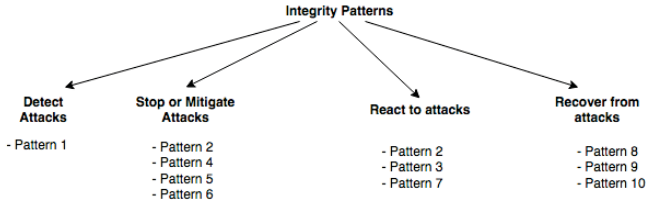

Fig. 1. Taxonomy of Integrity Patterns classified by “moments” (without catalog)

A pattern can be present in more than one category if it is capable of solving what is required at that moment. The patterns that are classified in the same category (“moment”) represent alternative solutions.

2) Usability: This QA represents all the design decisions that were considered to facilitate the user’s experience with the system. A system that exhibits usability requires implementing principles that prevent the user from making errors in the interaction as well as that the system has mechanisms to facilitate its operation.

The moments associated with the classification of usability patterns correspond to the categories of the taxonomy of Usability Tactics described in [15]. Figure 2 illustrates the taxonomy associated with Usability Patterns.

The stimulus associated with *Usability* is the interaction with the user. Then, there are two moments to consider:

- Support User Initiative: The system must provide the necessary mechanisms so that the user can see the status of his requests and be able to revert or manage the actions he performs.
- Support System Initiative: The system must be able to anticipate the user’s intention, predict certain actions according to the behavior of the system and provide mechanisms to prevent errors in the user’s interactions with the system.

D. Identify trade-offs

For illustrative purposes, let us analyze the impact that the selection of *Integrity* patterns has on other QA’s, according to a specific moment. This same analysis should be performed with the patterns related to the other QA’s to have a complete record of the associated trade-offs.

In Figure 1, we see that the ”stop or mitigate attacks” moment is satisfied using one of the four alternative patterns: Pattern 2, Pattern 4, Pattern 5, and Pattern 6. We will use a softgoals-based approach [6] to identify the trade-offs associated with each pattern. For each pattern, a ++ (make), −− (break), + (help) or − (hurt) will be selected.

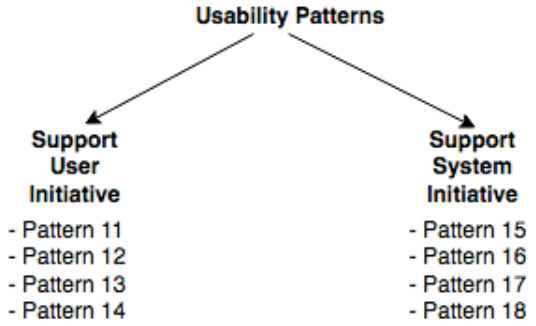

Fig. 2. Taxonomy of Usability Patterns classified by “moments” (without catalog)

In our example, patterns 5 and 6, are alternative solutions to a “moment” associated with a QA (Integrity). However, pattern 5 negatively impacts on another QA (Usability), while Pattern 6 has no impact on *Usability*. For this reason, we chose the selection of pattern 6 to be used in the design of the system.

Figure 3 corresponds to a Softgoal Interdependency Graph (SIG) [6] that represents the existing trade-offs between architectural patterns of a specific moment illustrating the impact of choosing one option over another in each QA that the system must satisfy.

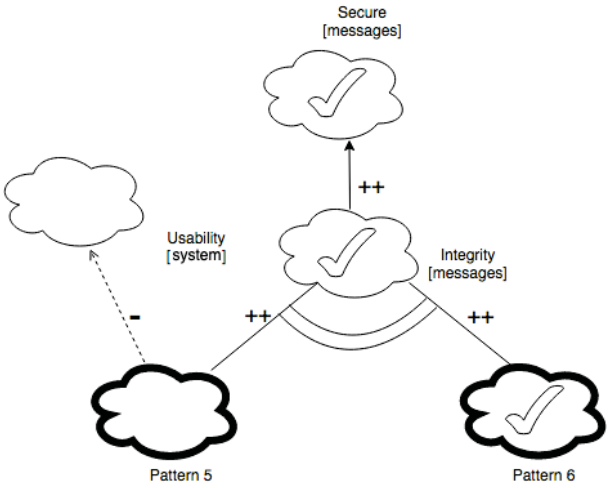

Fig. 3. SIG representing trade-offs between architectural patterns

By carrying out this process in a systematic way with the other patterns classified according to moments, it is possible to derive a subset of appropriate patterns for the system that must be built.

In our example we only consider a single case of trade-off identification to support the description of the step, but this trade-off analysis should never be considered in isolation since it would not be representative of the impact that all

the patterns have on the different QA's. For this reason, the analysis must be complete, about integrating and considering all the QA's associated with the system. This analysis will identify and evaluate the alternative patterns belonging to the same moment, and that qualify as more appropriate to be incorporated into the design of a system.

After performing a complete trade-off analysis, the following architectural patterns were selected to cover the needs of the system to be built:

$P = \{$*Pattern 1, Pattern 6, Pattern 7, Pattern 9, Pattern 11, Pattern 15*$\}$

IV. Illustrative Case

OmniSec is the internal name of a project that was designed and executed by our research group using our method. In this section, we will specify how it is possible to illustrate a way to apply our proposal to OmniSec. We will explore each step of our proposal, contextualizing it to the illustrative case.

A. Reviewing the business and architecture problem

In the fuel industry ecosystem, pipelines that transport several types of oil and gas use electronic sensors to keep track of the continuous flow of hydrocarbons. While the gas is transported and flowing through the pipelines, sensors are measuring the load that flows through them. The purpose of these sensors is to measure and provide data that facilitate the generation of reports and that serve as input for an operator who wants to monitor the accuracy and consistency of the flow historically.

The data obtained from these sensors are sent to an industrial flow computer, which can store a limited number of reports (up to 8 reports). Also, this flow computer provides functions such as sending a print job to a local or remote printer to have the reports printed.

An operator enters the data obtained from the printed report into an information system of the company to analyze and have persistence of these operational data over time. In this context of the operation, threats can be generated, such as the modification of the data by the operators or that the data are not available when required due to negligence or human inefficiency, among others.

The data obtained from the sensors have critical importance for the operation and commercial management of the industrial sector of fuels and gas; however, the technical limitations of these equipment suppresses the ability of the system for having historical operational data over time. Also, flow computers do not provide mechanisms that benefit or support the integration with third-party information systems.

To solve the problem described above, we designed a Cyber-Physical System (CPS) [19] that allows collecting, processing and saving the data provided by the sensors, and at the same time, exhibits functions such as integration with external systems and reporting the data to end-users.

From an architectural perspective, we identified the following QA's that the system must exhibit:

- **Integrity:** It must have mechanisms to ensure that the data associated with reports, configurations and operating system resources are not manipulated by unauthorized users, supporting and being in favor of the consistency of the stored data. It is required that communication channels cannot be intervened or manipulated.
- **Confidentiality:** Only authorized users should have access to read the resources that are provided by OmniSec (reports, configurations, and access to the operating system). The communication channels must guarantee that the data cannot be intercepted while it is being transmitted
- **Availability.** Allow the system to continue in operation in case of failure in the system at any of its software or hardware components.
- **Auditability.** The system should allow the reconstruction and supervision of both historical and ongoing events to authorized users.
- **Performance.** The system must react quickly to mitigate future threats without significantly impacting system resources.

B. Reducing options systematically

To perform this reduction, we selected the pattern catalog proposed by [2]. This set of patterns is appropriate for our analysis since: (1) there is evidence of the relevance and usefulness of these security patterns to design secure systems [2], and (2) specifically address Security.

The patterns described in the catalog are mentioned below:

- Application Firewall
- Audit Interceptor
- Authentication Enforcer
- Authorization Enforcer
- Checkpointed System
- Comparator Checked Fault Tolerant System
- Container Managed Security
- Controlled Object Factory
- Controlled Object Monitor
- Controlled Process Creator
- Credential Tokenizer
- Demilitarized Zone
- Encrypted Storage
- Firewall
- Full View with Errors
- Input Guard
- Limited View
- Load Balancer
- Obfuscated Transfer Object
- Output Guard
- Replicated System
- Reverse Proxy
- Secure Access Layer
- Secure Logger
- Secure Message Router
- Secure Pipe
- Secure Service Faade
- Secure Session Object
- Security Association
- Security Context
- Server Sandbox
- Session
- Session Failover
- Session Timeout
- Single Access Point
- Subject Descriptor

To filter this set of 36 initial patterns, we selected the most proper ones manually according to the characteristics and context for the domain of the problem. This process was validated by three senior architects of our research team.

As a result of this filtering process, the following set of patterns was obtained:

- P1: Application Firewall
- P2: Audit Interceptor
- P3: Authentication Enforcer
- P4: Authorization Enforcer
- P5: Comparator Checked Fault Tolerant System
- P6: Container Managed Security
- P7: Credential Tokenizer
- P8: Encrypted Storage
- P9: Firewall
- P10: Input Guard
- P11: Limited View
- P12: Output Guard
- P13: Replicated System
- P14: Secure Access Layer
- P15: Secure Logger
- P16: Secure Pipe
- P17: Secure Session Object
- P18: Server Sandbox
- P19: Session
- P20: Session Timeout

The output of the filtering process reduced the number of patterns from 36 to 20. With these 20 patterns, we used our systematic approach to classify them according to the categories associated with taxonomies of tactics described in the literature and later we identified the trade-offs between patterns belonging to the same moment concerning the QA's specified in this case.

C. Mapping patterns with QA's

We will illustrate how the selected subset of patterns can be classified according to the categories that are present in taxonomies of tactics for the desired QA.

To simplify our example, we will only illustrate the patterns related to *Confidentiality* and *Integrity*. Since these QA's are considered *Security* dimensions [23], we used the categories associated with the taxonomy of Security Tactics proposed in [24], to constitute the moments of our classification of architectural patterns related to both QA's. The moments associated with the taxonomy are the following: *Detect attacks*, *Stop or mitigate attacks*, *React to attacks*, *Recover from attacks*.

Figure 4 illustrates the taxonomy associated with Confidentiality Patterns. Each pattern is associated with one or more specific categories (moments).

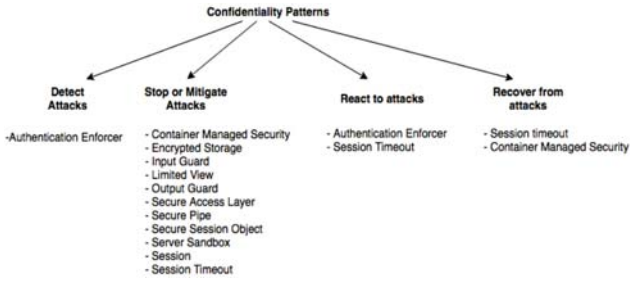

Fig. 4. Taxonomy of Confidentiality Patterns classified by "moments"

Figure 5 illustrates the taxonomy associated with Integrity Patterns.

Figures 4 and 5 classify the patterns according to the moments in which they have applicability. The patterns associated with each moment are alternative solutions.

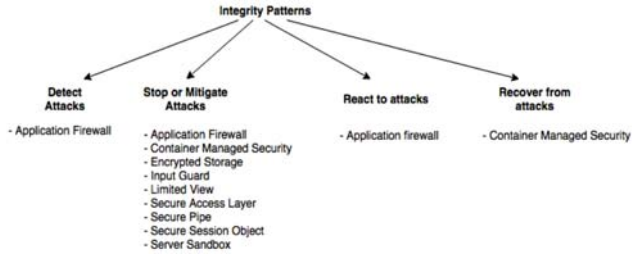

Fig. 5. Taxonomy of Integrity Patterns classified by "moments"

D. Identifying trade-offs

After building the taxonomy of patterns and classifying them according to moments, we proceed to identify the trade-offs that exist between them. Following the soft-goal-based trade-off representation and identification approach described in [6], we analyze the impact of the patterns belonging to the same moment regarding the QA's that the system must satisfy.

For each pattern, a ++ (make), −− (break), + (help) or − (hurt) was selected. The objective of this identification is to detect alternative solutions and being able to perform the decision making on choosing the best alternatives that cause positive effects and avoid the ones that can cause negative effects.

For example, consider the following scenario:

An attacker who has physical access to the private network performs a man−in−the−middle attack [17] to tamper an active session in the system.

To address this scenario from the architectural design point of view, we will select two Confidentiality Patterns (see Fig. 4) from the selected catalog: *Secure Pipe* and *Secure Access Layer*. Both patterns correspond to the "Stop or mitigate attacks" moment.

Figure 6 shows a Softgoal Interdependency Graph (SIG) [6], which illustrates the impact of choosing one or the other alternative solution. The main idea of this technique is to support the decision-making process by identifying trade-offs in the selection of patterns belonging to a specific moment.

Secure Access Layer promotes integrity in the transmission of messages, but negatively affects the performance of the system. *Secure Pipe* addresses *Integrity* without impacting the *Performance* of the system, so it seems an appropriate choice to solve the problem without negative consequences. To represent the impact of the other patterns on the QA's of the system, we elaborated a table that summarizes what was identified.

Table I illustrates the trade-offs identified for each pattern with respect to the QA's associated with the system.

From the previous analysis, the following pattern set was derived as the final selection that supported the design of the system:

$P = \{$*Application Firewall, Secure Pipe, Container Managed Security*$\}$

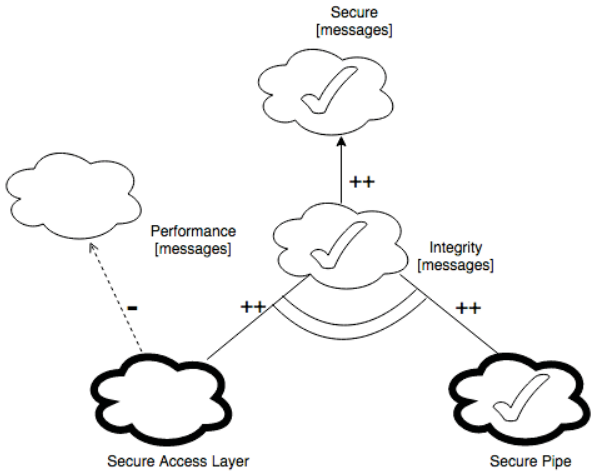

Fig. 6. OmniSec - Trade-off between architectural patterns

TABLE I
TRADE-OFFS BETWEEN ARCHITECTURAL PATTERNS

Pattern	Quality Attribute				
	Confidentiality	Integrity	Availability	Auditability	Performance
P1		++		–	
P2			++		
P3	++	+		++	–
P4		++		++	–
P5			++		–
P6	++	++			
P7	++				
P8	++	++			–
P9	+	+	++		–
P10	++	++	+	+	–
P11	++	++			
P12	++				–
P13		+	++	+	–
P14	++	++			
P15	+	+		++	–
P16	++	++			
P17	++	++			
P18	++	++	+		
P19	++			+	
P20	++				

These patterns were subsequently operationalized through technological implementations, which cannot be specified in detail due to non-disclosure agreements.

The proposed method was useful because it allowed us to consider different solution alternatives through a systematic approach that we will implement as part of a methodological framework in future developments.

V. DISCUSSION

For years, patterns have been the fundamental foundations for designing software architectures of all types of systems [25]. Similarly, tactics are architectural constructs that are directly related to the design decisions that a system must exhibit during its runtime [15]. We argue that the categorization of architectural tactics taxonomies can be used to classify architectural patterns under a similar taxonomy, where each category corresponds to the moment when the system manages an external stimulus.

Since each moment has a set of associated patterns, it is possible to compare patterns of the same moment with each other.

However, several cases can happen with this approach:

1) A moment has patterns that are complementary but independent of each other: If the patterns are complementary, then they are probably not alternative solutions, but as a whole, they are a solid solution.
2) A moment has complementary and highly coupled patterns: In this case, we can not perform a trade-offs analysis without considering the patterns together. The impact on other QA's corresponds to the impact of each of them separately, so when selecting a *Pattern A* highly coupled with a *Pattern B*, if *Pattern B* has a negative impact on a certain QA, when selecting *Pattern A*, the negative impact of *Pattern B* should be considered, regardless of whether *Pattern A* has no such impact.
3) A moment has mutually exclusive patterns: In this case, both patterns are alternative solutions. It is the best scenario to identify and analyze the associated trade-offs.
4) A moment has no associated patterns: In this case, likely, the selected pattern catalog is not complete enough to cover all the moments belonging to the taxonomy.
5) A moment has only one pattern: If there is only one pattern associated with a specific moment, then there is no possibility to perform a trade-offs analysis since there are no alternative solutions.
6) The same pattern belongs to different moments: If a pattern can belong to different times, it is necessary to analyze the trade-offs between alternative solutions for each moment to which the pattern belongs, because depending on the moment, the alternative solutions will be different.

Considering the above, the selection of an appropriate catalog of patterns is a key factor. The scenarios that make trade-off analysis complex can be mitigated through the incorporation of more than one catalog of patterns that allow a more significant number of alternative solutions, taking into account that these patterns must be comparable to each other. However, both the selection of the catalog and the subsequent identification of trade-offs between patterns is highly dependent on the experience of the analyst or architect of the system.

While we use the method in the context of one of our projects, and our team is composed of experienced software engineers and architects, we believe that the most useful of this proposal is related to its use by novice software engineers, Because there is evidence that pattern selection is a problem that mainly affects software engineers who do not have much experience [27].

We consider that development teams that use a traditional development process can benefit most than those who use an agile approach. This could be due to the lack of a tool that automates or supports the implementation of our method. We

need to carry out more studies that evaluate the relevance of our method in different types of software development processes.

VI. Conclusions and Future Work

In this article, we describe a method to classify patterns and subject them to evaluation based on criteria of positive, negative, or neutral impact concerning a QA that a system must achieve.

This analysis is only possible due to the previous classification of alternative patterns among themselves, whose classification criterion is based on existing categories in the context of architectural tactics. Based on these architectural constructs, we introduced the concept of "moment" to designate a category of alternative patterns that are executed before, during, or after an event (stimulus).

This method allows architects to have support for the decision-making process. The main objective of the method is to provide an approach in which architects can decide whether an alternative is appropriate or not based on the identification and selection of design decisions that respond to events that occur at specific times during the execution of a system.

As future work, our next step is to conduct experimental studies with practitioners to collect evidence about the usefulness and scope of our proposal in heterogeneous development teams. We are also interested in studying the applicability of our method to cases whose QA's are related to internal software properties; for example, modifiability, deployability, or testability. We expect to implement this method in a more automatic process by developing a recommender system [18] that allows supporting the pattern selection process under a moment-based approach.

Acknowledgment

This work has been partially supported by ANID through PCHA/Doctorado Nacional/2017-21171506 and PCHA/Doctorado Nacional/2017-21171351; ANID PIA/APOYO AFB180002; and by UTFSM through PIIC (Programa de Incentivos a la Iniciación Científica).

References

[1] L. Zhu, A. Aurum, I. Gorton, and R. Jeffery. "Tradeoff and Sensitivity Analysis in Software Architecture Evaluation Using Analytic Hierarchy Process," *Software Quality Journal*, December 2005, Volume 13, Issue 4, pp 357375.

[2] K. Yskout, R. Scandariato, and W. Joosen, "Do Security Patterns Really Help Designers?," *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Florence, 2015, pp. 292-302.

[3] K. Beck and R. E. Johnson. "Patterns Generate Architectures," *In Proceedings of the 8th European Conference on Object-Oriented Programming (ECOOP '94)*, 1994, London, UK, 139-149.

[4] A. Birukou. "A survey of existing approaches for pattern search and selection," *In Proceedings of the 15th European Conference on Pattern Languages of Programs (EuroPLoP '10)*. 2010. ACM, New York, NY, USA, Article 2, 13 pages.

[5] A. Ampatzoglou, G. Frantzeskou, and I. Stamelos. "A methodology to assess the impact of design patterns on software quality," 2012. *Inf. Softw. Technol. 54*, 4 (April 2012), 331-346.

[6] E. Yu. "Modelling strategic relationships for process reengineering." Ph.D. thesis, Department of Computer Science, University of Toronto, 1995.

[7] D. Gross and E. Yu. "From Non-Functional Requirements to Design through Patterns," *Requirements Engineering, February 2001*, Volume 6, Issue 1, pp 1836.

[8] N. B. Harrison, P. Avgeriou, and U. Zdun. "Using Patterns to Capture Architectural Decisions," *2007, IEEE Softw. 24, 4 (July 2007)*, 38-45.

[9] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, and M. A. Babar. "A comparative study of architecture knowledge management tools," Mar-2010, *In Journal of Systems and Software. 83*, 3, p. 352-370 19 p.

[10] M. Fowler. *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[11] E. Fernandez. *Security Patterns in Practice: Designing Secure Architectures*. John Wiley and Sons Inc. in 2013.

[12] A. Motii, B. Hamid, A. Lanusse, and J. Bruel. "Guiding the selection of security patterns based on security requirements and pattern classification," *2015, In Proceedings of the 20th European Conference on Pattern Languages of Programs (EuroPLoP '15)*. ACM, New York, NY, USA, Article 10, 17 pages.

[13] H. Shahid, K. Jacky, K. A. Ali, B. K. Ebo. "A Methodology to Automate the Selection of Design Patterns," *In proceedings of 2016 IEEE 40th Annual Computer Software and Applications Conference Workshops*. Vol. 2 IEEE Computer Society, 2016. p. 161-166.

[14] J. Yoder and J. Barcalow. "Architectural Patterns for Enabling Application Security," *In Information Systems Security 12,* November 1998.

[15] L. Bass, P. Clements and R. Kazman. *Software Architecture In Practice: Third Edition*. Addison-Wesley Professional; 3 edition (October 5, 2012).

[16] T. Saaty. "Decision making with the analytic hierarchy process," *Int. J. Services Sciences,* Vol. 1, No. 1, 2008.

[17] M. Conti, D. Nicola, and L. Viktor. "A Survey of Man In The Middle Attacks", in IEEE Communications Surveys and Tutorials, vol. 18, no. 3, pp. 2027-2051, third quarter of 2016.

[18] F. Ricci, L. Rokach, and B. Shapira. *Recommender Systems: Introduction and Challenges*, 2015.

[19] R. H. Rawung and A. G. Putrada. "Cyber-physical system: Paper survey," *2014 International Conference on ICT For Smart Society (ICISS)*, Bandung, 2014.

[20] M. Makki, E. Bagheri, and A. A. Ghorbani, "Automating architecture trade-off decision making through a complex multi-attribute decision process", in *Software Architecture*. New York, NY, USA: SpringerVerlag, 2008, pp. 264272

[21] F. Buchmann and L. Bass. "Introduction to the attribute driven design method," *Proceedings of the 23rd International Conference on Software Engineering. IEEE Computer Society*, Toronto, Ontario, Canada (2001).

[22] M. Salama, R. Bahsoon, and N. Bencomo, "Managing trade-offs in self-adaptive software architectures: A systematic mapping study" *in Managing trade-offs in adaptable software architectures*, Eds. Elsevier, 2016.

[23] W. Stallings and L. Brown, *Computer security: principles and practice.* New York, NY: Pearson, 2018.

[24] E. B. Fernandez, H. Astudillo, and G. Pedraza-García "Revisiting architectural tactics for security" *9th European Conference on Software Architecture (ECSA)* pp. 55-69 2015.

[25] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design patterns: Elements of reusable object-oriented software.* Reading, MA: Addison-Wesley, 1995.

[26] V. Gruhn, and R. Striemer, *The Essence of Software Engineering*, Springer, 2018.

[27] E. M. Sahly and O. M. Sallabi, "Design pattern selection: A solution strategy method," *2012 International Conference on Computer Systems and Industrial Informatics*, Sharjah, 2012, pp. 1-6.

[28] I. Sommerville, *Software Engineering (10th Edition)*, Pearson, 2016.

[29] S. Hussain, J. Keung, M. K. Sohail, A. A. Khan and M. Ilahi, "Automated framework for classification and selection of software design patterns", Volume 75, 2019, Pages 1-20, ISSN 1568-4946.