Modeling and Selecting Frameworks in terms of Patterns, Tactics, and System Qualities

Hind Milhem*, Michael Weiss, Stephane S. Somé*
*School of Electrical Engineering and Computer Science (EECS), Department of Systems and Computer Engineering
*University of Ottawa, Carleton University
Ottawa, Canada
hbani043@uottawa.ca, michael_weiss@carleton.ca, ssome@eecs.uottawa.ca

***Abstract*—Selecting frameworks and documenting the rationale for the choice is an essential task for system architects. Different framework selection approaches have been proposed. However, none of these connects frameworks to qualities based on their implemented patterns and tactics. In this paper, we propose a way to compare automatically the quality attributes of frameworks by extracting the patterns and tactics from a framework's source code and documenting them to connect frameworks to requirements upon which a selection can be made. We use a tool called Archie (a tool used to extract tactics from a Java-based system's code) to extract the patterns/tactics from the implementation code of frameworks. We then document and model these patterns/tactics and their impact on qualities using the Goal-oriented Requirements Language (GRL). The satisfaction level of the quality requirements integrated with other criteria such as the preferences of an architect provide architects with a tool for comparing different frameworks and documenting their rationale for choosing a framework. As a validation of the approach, we apply it to realistic case studies with promising results.**

Keywords-Framework Selection; Architectural Tactic; Architectural Pattern; Non-Functional Requirement (NFR); Framework Modeling; Tactic/Pattern Extraction

I. INTRODUCTION

A framework is a highly reusable design for an application or part of an application in a given domain. With the increasing complexity of developing software systems and shorter delivery times, it is essential to reuse existing designs in the form of frameworks as much as possible. Many candidate frameworks are usually available for a given application. Therefore, selecting frameworks and documenting the underlying rationale for the choice, become an important task for system architects. Various previous work [1][2][3] have addressed the selection of frameworks based on various characteristics and criteria such as the features of the frameworks, the deployability and the interoperability of the frameworks, and how they perform (testing). However, none of these connects frameworks to qualities to compare frameworks based on their exhibited quality attributes expressed as Non-Functional Requirements (NFRs) [4].

DOI reference number: 10.18293/SEKE2020-049

Architectural patterns and tactics [4] are reusable building blocks for software development (including frameworks). They are characterized in terms of factors that affect the various quality attributes so that architecture can be understood in terms of those quality attributes. Our main assumption is that the implementation of a framework inherits from the quality attributes associated to the patterns and tactics used in its implementation.

In a previous work [5], we proposed an approach to select frameworks based on their quality attributes. We associate frameworks with quality attributes based on the architectural patterns used in their implementation. Since the implementation of frameworks is not always adequately documented, we use a source code analysis tool (Archie [6][7][8]) to determine which architectural patterns are used in the implementation of the frameworks. We then model the relation between the frameworks, patterns and quality a Goal model using the Goal-oriented Requirements Language (GRL) [9].

This work builds on our prior work [5] by considering two additional characteristics to select a framework in addition to architectural patterns. This is because selecting a framework based only on its patterns might not be sufficient. The two additional characteristics are the architectural tactics and the importance values of quality requirements (preferences of architects). We use the Archie tool to find the tactics implemented in frameworks. We then add the tactics and their impact on quality attributes to the GRL model. We calculate the importance values of the NFRs using the AHP method [10]. Adding the implemented tactics and the importance values of the NFRs as other criteria to the model, in addition to the patterns, would provide more details about how the implemented patterns/tactics, considering the architects' preferences in a framework, can together push or pull the framework toward or away from given NFRs in an informed way.

The remainder of this paper is organized as follows; Section 2 provides an overview of the related work. We present our proposed approach in Section 3. In Sections 4, we present a case study as an example to apply the approach to. Section 5 shows the preliminary validation of the approach. In Section 6, we

present threats to the validity of this work. In Section 7, we draw initial conclusions and describes plans for future work.

II. RELATED WORK

Cervantes et al. [1] extract patterns and tactics from a framework by applying a mapping process between the patterns and tactics in a framework and those patterns and tactics, which are employed in architecture design. They also mention that patterns can be extracted from the provided services of a framework and that framework selection is based on architecture drivers (such as the team's level of knowledge of a framework, or the framework's maturity). In comparison to our work, patterns are identified manually. This approach does not consider the patterns and tactics implemented in a framework as a selection criterion. It also does not provide any details on how to represent frameworks in terms of the patterns and tactics they implement.

Mirakhorli and Huang [6][7][8] present an approach that relies primarily on information retrieval and machine learning techniques for discovering tactics in code. This is done by training a classifier to recognize specific terms that occur commonly across implemented tactics. The probabilities of these terms (the probability that a particular term identifies a class associated with a tactic) are determined using specific mathematical equations. The resulting tool is called Archie and is used in our work to identify architectural patterns and tactics from source code. In comparison to our work, their work only focuses on the tactics. Their work does not focus on the selection method and modeling of architectures in terms of their implemented patterns and tactics.

Sena et al. [12] analyze studies reporting on software architectures of big data systems, to identify architectural patterns, quality attributes, as well as problems and liabilities of those patterns. They determined that various architectural patterns, such as the Layered pattern, the Pipe and Filter pattern, the Broker pattern, and the Shared Repository pattern have significant impacts on the qualities and characteristics of big data systems. We use the results of this work to determine the main quality requirements and the determined patterns, as discussed in the technical report [11]. In comparison to our work, this work does not focus on the modeling of architectures in terms of their implemented patterns. The extraction of the patterns is done manually by analyzing studies reporting on software architectures of big data systems.

Additional related work includes: Johnson [13], Aguiar and David [14], Beck and Johnson [25], Ryoo et al. [29], and Meusel et al. [30]. A key difference between our work and these is that their work does not connect frameworks to quality attributes based on both patterns and tactics. Their work also does not focus on the selection method and modeling of architectures in terms of their implemented patterns and tactics.

III. PROPOSED APPROACH

The proposed approach includes three general steps: First, determining the patterns and tactics a framework implements. Second, modeling the frameworks in terms of their patterns and tactics. Third, choosing a framework.

In the following, we present the general steps (process) of the approach.

A. Determining Patterns and Tactics Implemented in a Framework

To determine the implemented patterns and tactics of a framework, we follow the following sub-steps:

1) Determine the Context/Domain

The objective of this sub-step is to restrict the scope of the search. This allows a more focused determination of the candidate frameworks, the patterns, and the tactics according to a specific context.

2) Choose the Patterns and Tactics that Need to be Checked for A Framework in the Determined Context/Domain

The set of patterns and tactics applied to a problem is typically restricted by the domain and context of that problem. These patterns and tactics are the ones known to contribute to solving aspects of the problem. Our approach searches for the patterns and tactics relevant to the context of a problem in frameworks used as part of the solution to this problem. Therefore, a knowledge of these patterns and tactics is needed as input.

In this work, we conduct a literature review to find the most relevant and common patterns and tactics of a framework in the determined context. The resulting list of tactics and patterns is however reusable in the same domain.

3) Determine the Tool to be Used to Extract the Patterns and Tactics of A Framework

Although a manual search of the implemented patterns and tactics in a framework is possible, it is not practical for large frameworks. Different alternative methods have been used in the literature such as Archie [6][7][8], Matching methods between the provided services of a framework and its patterns/tactics [13], Pattern instantiation (assigning the roles defined in a pattern to concrete classes, responsibilities, methods, and attributes of a practical design) [14], and Matching methods between the problem statement of an architecture and the applied patterns [15].

In this work, we use Archie [6][7][8] to extract the patterns and tactics from the frameworks' source code. We chose Archie because it is the only automated tool among the alternatives. So, it makes the extraction process faster by decreasing time and effort spent searching the patterns and tactics and their related terms in the documentation, websites, and source codes of frameworks. It is extensible so we can add or remove patterns and tactics.

4) Apply the Tool on a Framework to Extract the Patterns and Tactics it Implements

In this sub-step, we apply Archie on the candidate frameworks and get a set of candidate patterns and tactics for each framework. The interested reader may find more details about this step in [11].

5) Validate the Candidate Set of Patterns and Tactics which are Detected by the Selected Tool

We validate the results of applying Archie on the candidate frameworks by looking for the occurrences of those patterns/tactics, which are detected by Archie, manually in the

source code/documentation/websites of the candidate frameworks. The goal of this step is to ensure the validity of our results. For more details about this step, see [11].

B. Modeling Frameworks in terms of their Implemented Patterns and Tactics

To model frameworks in terms of their implemented patterns and tactics, we perform the following sub-steps:

1) Determining the Modeling Language to be Used to Model Frameworks

Different modeling languages have been used to model frameworks. Examples include The Goal-oriented Requirement Language (GRL) [9], the NFR-framework [16], i* (i-star) framework [17], and the softgoal modeling language [18].

In this work, we chose the GRL. The elements of the GRL notation used are shown in Figure 1. The choice of GRL was motivated by the facts that: it enables us to evaluate and compare the impact of different design choices on quality attributes, it is a part of an international standard (User Requirements Notation – URN) [9], enables the modeling of stakeholders and their goals, supports Key Performance Indicators (KPIs) for quantitative reasoning, and supports evaluation strategies and propagation algorithms to evaluate the satisfaction of goals and actors under selected conditions [19]. Giving quantitative contributions of patterns and tactics helped us calculate the satisfaction of NFRs.

2) Modeling the Patterns, Tactics, and their Contributions on the NFRs

In this sub-step, we first extract from the description of the patterns/tactics, the NFRs, the contributions of the patterns and tactics on the NFRs. Then, we extract the design decisions, which show the reason for the negative or the positive impact of a pattern/tactic on an NFR. In this work, we follow Ong et al.'s [20] approach to extract NFRs, design decisions, and the contributions of the patterns and tactics on the NFRs. We added to the description by underlining the benefits, liabilities, the affected NFRs, and reasons for the positive or negative impact of the patterns or tactics on the NFRs.

The benefits and liabilities of a pattern/tactic indicate the positive and negative contributions on the NFRs respectively. The reasons for the positive or negative impact of the patterns/tactics on the NFRs correspond to design decisions behind the application of a pattern/tactic. These design decisions are expressed as sentences starting with an active verb such as 'define,' 'register,' 'change,' 'reuse,' etc. We also have followed the same method for the tactics.

We then derive GRL models, with the NFRs and the contributions of the patterns and tactics on the NFRs, from the description of each pattern/tactic. First, we start with the patterns/tactics at the bottom of the model. Then, we put the design decisions and NFRs at the topmost level of the model. The complexity of the system dictates the number of levels of design decisions.

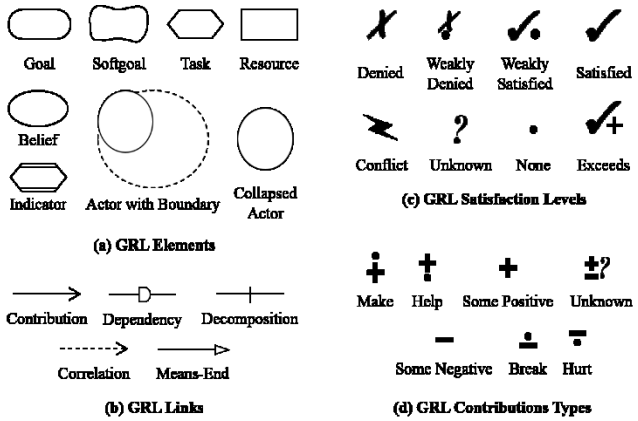

Figure 1. Summary of the GRL notations [9]

Based on Figure 1, we select softgoals (clouds) elements to represent NFRs and the design decisions, indicating that these cannot be achieved in an absolute manner. Tasks (hexagons) are selected to represent patterns, tactics, the parts of a framework where a pattern/tactic is implemented, and frameworks, representing ways of achieving a softgoal. An actor with Boundary (dotted circle) is used to represent an architect of a framework. Solid lines (Contribution links) indicate the desired impacts of one element on another element. Contribution types determined by labels. These labels indicate various degrees of positive (+) or negative (-) contributions (see Figure 2 for the complete set of labels). Decomposition links allow an element to be decomposed into sub-elements [9]. AND, IOR and XOR are supported decompositions. We use only AND decomposition links to represent the connection between a framework and its patterns and tactics because all the patterns are required in a framework before the NFRs are satisfied. We used it also to represent the connection between the parts of a framework and the patterns and tactics because all the patterns and tactics are needed to be implemented in a part of a framework.

We use quantitative contribution values. There are different methods to get the contribution values of a pattern/tactic to an NFR such as AHP [10], Delphi [21], or by using indicators (one of the GRL notations as we can see in Figure 1). We use a matching method between the contribution between a pattern/tactic and a given NFR from the literature [22][23][24][25][26][27][28] and the contribution values used in the GRL. More details about the calculations of the contribution values are shown in [11].

3) Modeling Framework in terms of their Implemented Patterns and Tactics

The GRL models of the patterns and tactics from the previous sub-step, are used to build a bottom up GRL model for frameworks, starting with the framework and its parts at the bottom level of the model, connected with all its implemented patterns and tactics. The parts of a framework show where its patterns and tactics are implemented. A link between a pattern and a tactic indicates that the tactic is used as part of the pattern implementation. The resulting GRL model specifies that the

design decisions explain why a pattern/tactic impacts an NFR the way it does. Consequently, the design decisions push or pull the framework towards or away from NFRs, as shown in Figure 2.

Each NFR is assigned an importance value given by architects to help compare and choose the best suited framework. We calculate these importance values using the AHP method, as shown in the [11].

C. Choosing a Framework

1) Evaluate the models of the candidate frameworks

To initially assign a satisfaction level to a pattern/tactic, we assign a tactic or a pattern to be Satisfied (100) if a framework implements a tactic or a pattern; else, if a framework does not implement it, it is then assigned to be Denied (0). The initial values are marked with a star (*) on the evaluation model. All the patterns and tactics, which are implemented in a framework, are initially assigned using a star (*). After the initial assignment of satisfaction levels to the tactics and patterns of a framework, we evaluate the satisfaction levels of the NFRs by applying different evaluation strategies on the GRL models, as we will see in Section IV(C).

2) Compare the Candidate Frameworks

In this last step, we compare the candidate frameworks based on their implemented patterns and tactics considering the importance values of the NFRs, which would be given by an architect, as we will see in Section IV(C).

IV. CASE STUDY

To validate the approach, we applied our approach to an industrial case study, which is a part of a project to develop a cyber fusion center. The case study consists in choosing a stream processing framework for big data. Architects had to choose among different candidate frameworks. The selected framework was to provide the backbone for the collection and correlation of security events. Processing the events requires routing information from sensors to various processing stages that perform analytics on the events at different levels of abstraction (such as detecting attacks and attack patterns).

Our industrial collaborators considered three candidate frameworks: Apache Storm [29] (a component in Apache Metron [30]), Apache Flink [31], and Apache Spark [32]. In the following, we apply the main steps of our approach.

A. Determining Patterns and Tactics Implemented in a Framework

We apply the following sub-steps to determine the implemented patterns and tactics of the frameworks Apache Storm, Apache Flink, and Apache Spark.

1) Determine the Context/Domain

We determined the context of this project to be as big data systems in general and data streaming frameworks in specific. All the candidate frameworks are real data streaming frameworks.

2) Choose the Patterns and Tactics that Need to be Checked for A Framework in the Determined Context/Domain

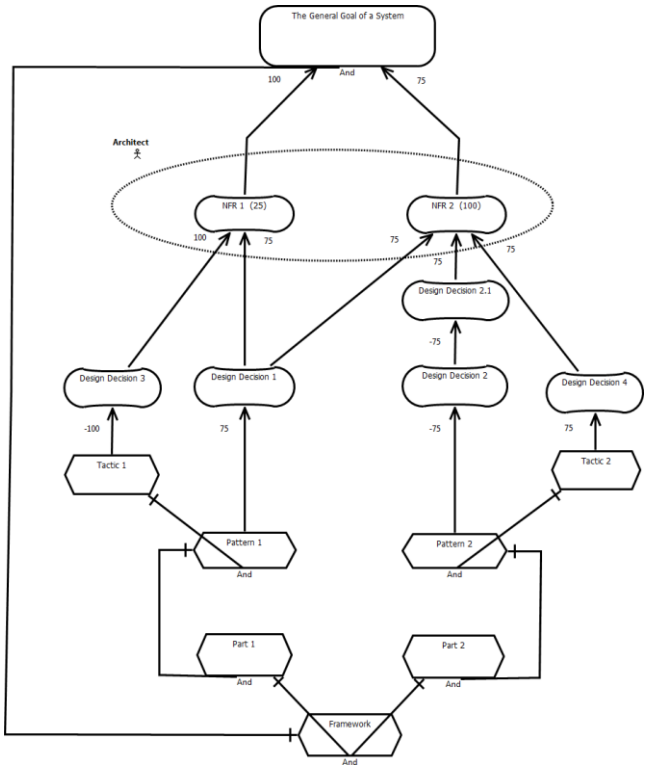

Figure 2. The general GRL model of a framework

To perform this sub-step, we conducted a literature review to find the most relevant and common patterns and tactics of a framework in the determined context.

Given the context of the problem, we conducted a literature review to find the most relevant patterns and tactics of a big data system in general and a data streaming system in specific. We also determine the most common NFRs of a data streaming framework. The results of this step and more details are shown in [11].

3) Determine the Tool to be Used to Extract the Patterns and Tactics of a Framework

We use the Archie tool [6][7][8] to extract the patterns and tactics from the frameworks source code as discussed in [7].

4) Apply the Tool on A Framework to Extract the Patterns and Tactics it Implement

Mirakhorli and Huang [6][7][8] trained a classifier in Archie to recognize specific terms that occur commonly across implemented tactics and calculate the weights of the tactics (the probability that a particular term identifies a class associated with a tactic). Archie tool considers thirteen tactics [6][7][8] from three quality attributes to be detected in any Java-based system. These tactics are *Policy-Based Access Control (PBAC), Role-Based Access Control (RBAC), Kerberos, Audit trail, Session Management, and Authenticate* from Security, *Checkpoint, Heartbeat, Ping/Echo, Active Redundancy, and Load Balancing* form Reliability, *and Resource Scheduling, and Resource Pooling* from Performance.

In addition to these thirteen tactics, we added seven other tactics and five patterns to be detected by the Archie tool. To see the added patterns and tactics, we refer to [11]. The analysis of the results of applying Archie to Storm, Flink, and Spark is shown in [11].

5) Validate the Candidate Set of Patterns and Tactics which are Detected by the Selected Tool

After applying the tool on the candidate frameworks, we validated the results by looking for the occurrences of the detected patterns/tactics, manually in the source code/documentation/websites of the candidate frameworks Storm, Flink, and Spark. The sample results of the validation are shown in [11].

B. Modeling Frameworks in terms of their Implemented Patterns and Tactics

We modeled the candidate frameworks Storm, Flink, and Spark in terms of their implemented patterns and tactics following the general model shown in Figure 2. The case study considers NFRs relevant to data streaming systems such as *Scalability, Maintainability, Performance, Portability, Availability, Reliability, Security, and Interoperability*. For the sake of readability, the presented model in Figure 3 is restricted to *Testability, Security, Reliability, Availability, and Scalability*. The high-level goal of the project is shown at the top of the model connected to alternative candidate frameworks at the bottom of the model. On top of each framework, there are several parts for each framework connected to their implemented patterns and tactics. The design decisions explain why a pattern/tactic impacts an NFR the way it does at the top of the model. Consequently, the design decisions push or pull the framework toward or away from NFRs.

C. Choosing a framework

1) Evaluate the models of the candidate frameworks

We evaluate the model to calculate the satisfaction levels of the NFRS (Figure 3). The evaluation is done by applying different evaluation strategies on the GRL model. For example, Figure 4 shows a first strategy where only the patterns and tactics implemented in the Spark framework are initially satisfied. Similarly, Figures 5 and 6 show strategies where only the patterns and tactics implemented in Flink and Storm, are initially satisfied. Color-coding is used to highlight what is satisfied and what is denied. For example, the 'Green' colour indicates that the element is satisfied, while the 'Yellow' colour indicates that the element is neutral. The 'Red' colour indicates that the element is denied.

2) Compare the candidate frameworks

Based on the evaluation results of the GRL models from the previous sub-step, we can see that the three frameworks have similar satisfaction levels of the *Testability, Security, and Scalability* requirements as shown in Figures 4, 5, and 6. The *Testability* requirement is satisfied with (42) satisfaction level for all the frameworks. While the *Security* is satisfied with (50) satisfaction level and *Scalability* with (56) satisfaction level for all the frameworks.

The Storm framework has a higher satisfaction level for *Reliability* and *Availability*, which is (63) compared to Spark and Flink. This is because of the implementation of the three

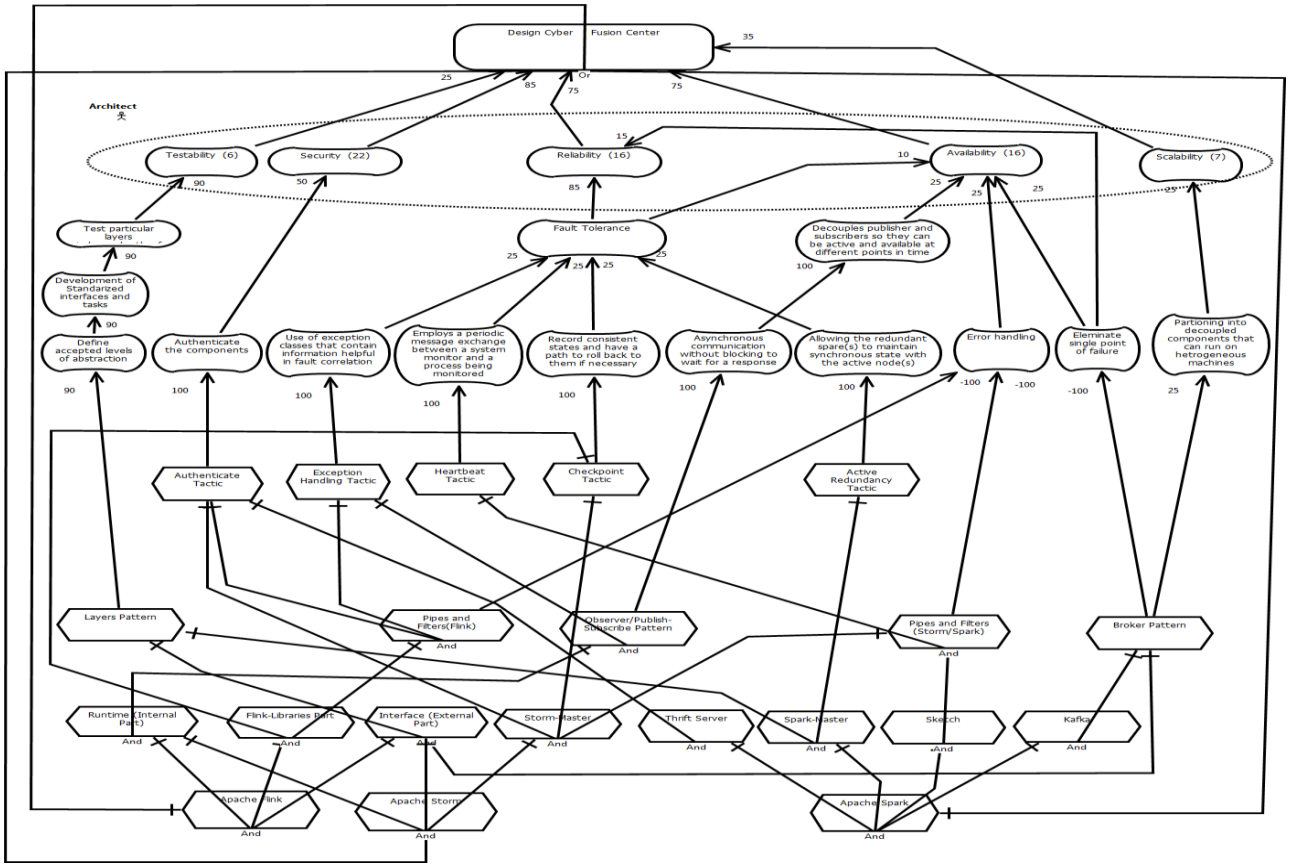

Figure 3. The GRL model of the Storm, Flink, and Spark frameworks in terms of Testability, Security, Reliability, Availability, and Scalability

reliability tactics: *Exception Handling, Heartbeat,* and *Checkpoint*. They all improve fault tolerance, which improves reliability. Spark and Flink have the same satisfaction level for Reliability, which is (42). Spark has the least satisfaction level for *Availability, which* is (5). While Storm has (32) and Flink has (30) satisfaction levels for *Availability*. This is because of applying the Observer/Publish-Subscribe pattern in Storm and Flink, which provides Asynchronous communication between components without blocking to wait for a response. This helps decouple publishers and subscribers so they can be active and available at different points in time, resulting in improving the availability of the frameworks. Both Storm and Flink use the "Checkpoint" tactic to Record consistent states and have a path to roll back to them if necessary. While Spark uses the "Active Redundancy" tactic for recovery, preparation, and repair of the errors. The architect is more satisfied with Storm than Flink and Spark. As we see, the satisfaction value of the architect for Storm is (48), while it is (43) for Flink and (37) for Spark. If an architect favours Reliability and Availability over the other requirements, we recommend Storm. However, if Testability, Security, and Scalability are preferred, then any one of the three frameworks could be equally recommended.

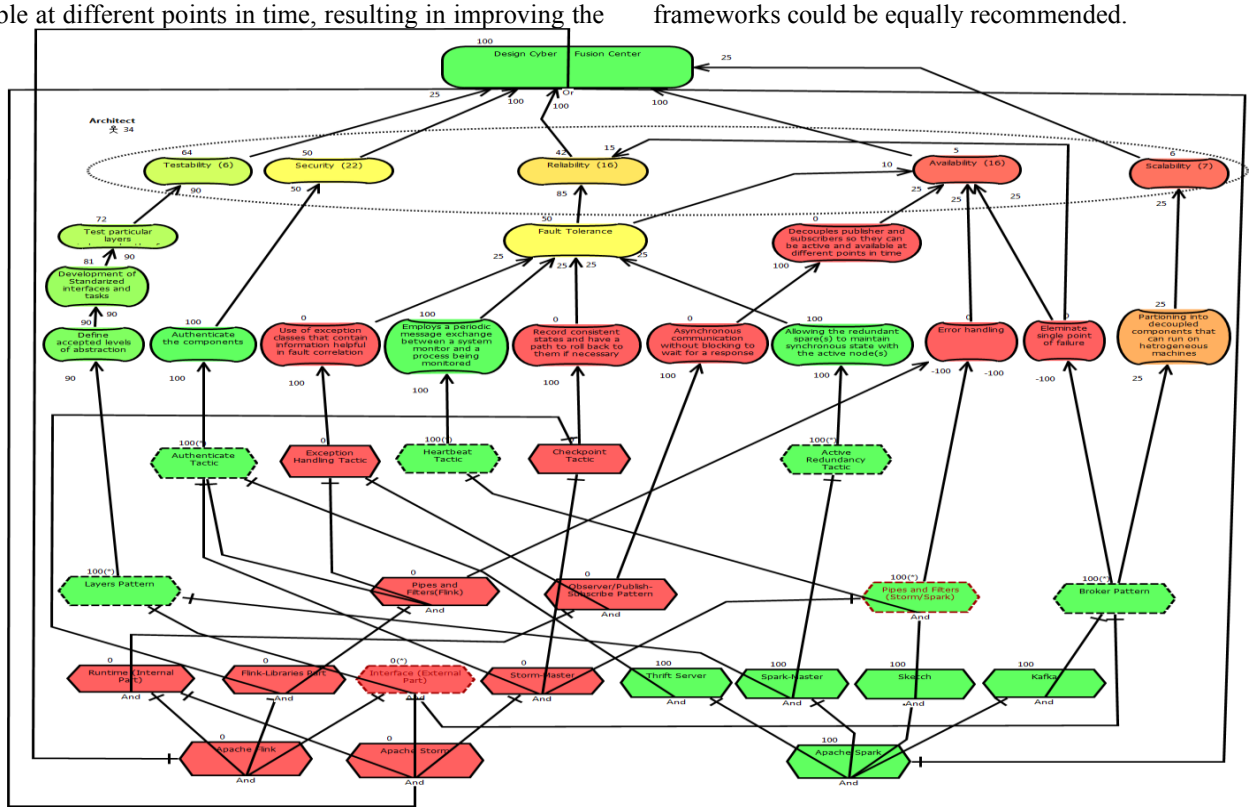

Figure 4. Strategy 1: Applying only the implemented patterns and tactics of the Spark

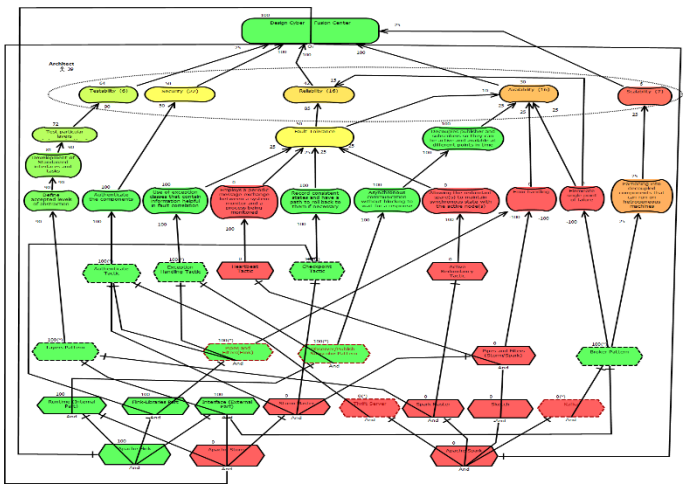

Figure 5. Strategy 2: Applying only the implemented patterns and tactics of the Flink

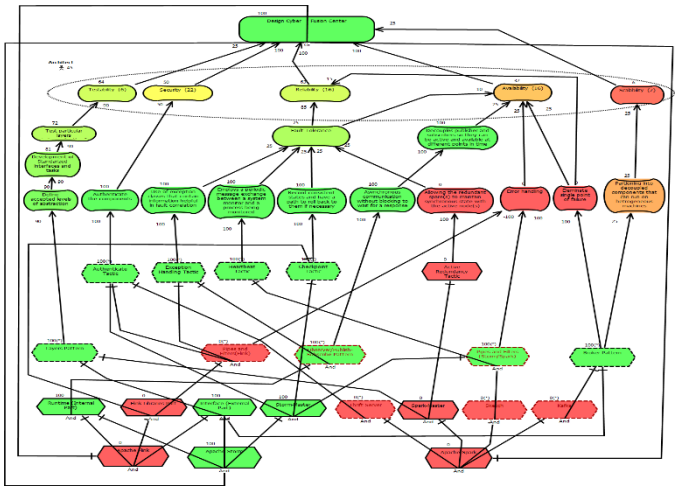

Figure 6. Strategy 3: Applying only the implemented patterns and tactics of the Storm

V. PRELIMINARY VALIDATION

The previous sections discuss the application of the approach to a case study. We applied the approach in the context of an industrial project where architects had to choose among different frameworks Spark, Storm, and Flink. The results were found satisfactory (and in agreement) with the project architects. The architects confirmed that the approach was helpful in choosing the best-fit framework to provide the backbone for the collection and correlation of security events in a cyber security center. We also compared the inferred quality attributes (i.e. reliability, availability, and performance) with benchmark comparison results such as [33]. Inoubli [33] showed that both Storm and Flink use the "Checkpoint" tactic for fault tolerance. While Spark uses recovery techniques. This was compatible with our results in Section IV(C). Our results showed that both Storm and Flink implement the "Checkpoint" tactic to Record consistent states and have a path to roll back to them if necessary. While Spark uses the "Active Redundancy" tactic for recovery, preparation, and repair of errors. Inoubli also showed that Spark is the fastest framework in terms of the processing time compared to Storm and Flink. This was compatible with our results, which shown in [11], that the satisfaction level of the Performance for Spark is (86) while it is (46) for both Storm and Flink. This confirms that Spark is the fastest one while Strom and Flink are quite similar in terms of the data processing speed, as shown in Figures 12 and 13 in Section IV(C).

Inoubli also reported that Flink and Storm share similarities and characteristics with Spark. Flink, Storm, and Spark implement similar patterns, such as the Layers and Broker patterns and similar tactics, such as "Resource Pooling" and "Resource Scheduling". The compatibility with Inoubli's results offers some validation of the main tenet of our works; the link between the implemented patterns and tactics, and quality attributes.

In another case study on Gradle and Maven tools [34], we also compared the inferred quality attributes (i.e. performance) with benchmark comparison results such as [34]. The results of the experiment conducted in [34], showed that Gradle is faster than Maven. This is because of the performance features, which Gradle includes, such as the parallelism and the incremental build and subtasks. In our results, which are shown in [11], we got quite similar results to the ones in [34].

In a case study on a Healthcare-Supportive System-System of Systems (HSH-SoS) architecture [35], we use our approach to support an analysis of the HSH-SoS architecture in terms of its implemented patterns and tactics. Our objective is to confirm that the approach can be used not only to compare implementations but also to provide a rationale or documentation about a framework/system architecture.

I. THREATS TO VALIDITY

Threats to validity can be classified as construct, internal, and external validity. We discuss the threats, which potentially impact our work, and the ways in which we attempted to mitigate them.

External Validity evaluates the generalizability of the approach. The primary threat is related to the assumption that a framework inherits the aspects of quality associated to its implemented tactics/patterns. It is possible that patterns/tactics could be implemented the wrong way and not provide their expected benefits. Although our initial validation with case studies such as Gradle and Maven has showed the validity of our assumption, more case studies will however be required. As mitigation to this threat we confirmed the proper implementation by performing a manual inspection of the code. Another threat is that NFRs derived from patterns/tactics such as performance might not be sufficient to be able to compare the frameworks. We consider the result provided by our approach as one component of the criteria for a final decision on choosing a framework. Other criteria including the cost, stability, maturity, community support might also be considered.

Construct Validity evaluates the degree to which Archie was accurate in detecting the patterns and tactics of the frameworks. In our case study, we have calculated the false positives and false negatives numbers by checking if those patterns/tactics detected by Archie are implemented in the source code of a framework. We found that there were only 12% false positives in Storm, 16% in Flink, 4% in Spark, and 16% in both Gradle and Maven. The whole results showed that most of the patterns and tactics, which were detected by Archie for the frameworks, are implemented in the frameworks. This confirms the high accuracy and performance of the Archie tool. Archie also has been tested on several systems ranging from 1,000 to 20,000 java files [6][7].

Internal Validity reflects the extent to which a work minimizes systematic error or bias so that a causal conclusion can be drawn. A threat to validity is that the search for specific patterns or tactics was solely performed by the authors. In the case of the cyber fusion center project, we mitigated this threat by elicited feedback from developers and architects with extensive experience with the involved frameworks.

VI. CONCLUSION AND FUTURE WORK

The approach described in this paper extracts the implemented architectural patterns and tactics from frameworks source codes to connect frameworks to quality requirements upon which a selection can be made. We use an information retrieval approach, with a tool called Archie, to determine the implemented architectural patterns and tactics in order to enable a more informed assessment by architects. We then model the frameworks in terms of their implemented patterns and tactics using the Goal-oriented Requirements Language (GRL). This model provides architects with a rationale about the satisfaction levels and the analysis of the tradeoff of given NFRs for a framework. Providing such rationale with considering the importance values of the NFRs integrated with other criteria such as the cost, delivery time,

stability, and maturity of a framework would help an architect to choose among several candidate frameworks.

In the future, we plan to improve our modeling of frameworks with GRL indicators instead of simply matching the impact of patterns on NFRs and the contribution values in the GRL. The indicators in the GRL measure observable values and convert them to GRL satisfaction values (from zero for denied, to 100 for satisfied) that can be propagated to other model elements through links. This would allow getting the contribution values of the patterns and tactics automatically.

Another future work is to integrate the consideration of criteria such as cost, delivery time, stability, and maturity of a framework in addition to the patterns, tactics, and the importance values of the NFRs to be able to choose a framework in a more informed way.

REFERENCES

[1] H. Cervantes, P. V. Elizondo, and R. Kazman. 2013. A principled way to use frameworks in architecture design. *IEEE Software*, March/April, 46-53.

[2] G. Grau, and X. Franch. 2007. A Goal-Oriented Approach for the Generation and Evaluation of Alternative Architectures. *European Conference on Software Architecture (ECSA)*, pp 139-155.

[3] A. Zalewski. 2013. Modeling and Evaluation of Software Architecture. *Warsaw University of Technology Publishing Office*.

[4] L. Bass, P. Clements, and R. Kazman. 2012. Software Architecture in Practice. *Addison-Wesley*.

[5] H. Milhem, M. Weiss, and S. Some. 2019. Extraction of Architectural Patterns from Frameworks and Modeling their Contributions to Qualities. *Pattern Languages of Programs (PLoP)*. 17 pages, Ottawa, Canada.

[6] M. Mirakhorli. 2014. Preserving the Quality of Architectural Tactics in Source Code.

[7] M. Mirakhorli and J. Cleland-Huang. 2016. Detecting, Tracing, and Monitoring Architectural Tactics in Code. *IEEE Transactions on Software Engineering*, Volume: 42, Issue 3, pp 205-220.

[8] M. Mirakhorli, A. Fakhry, A. Grecho, M. Wieloch, and J. Cleland-Huang. 2014. Archie: A Tool for Detecting, Monitoring, and Preserving Architecturally Significant Code. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp 739-742, Hong Kong, China.

[9] G. Mussbacher, M. Weiss, and D. Amyot. 2007. Formalizing Architectural Patterns with the Goal-oriented Requirement Language. *Proceedings of the Fifth Nordic Conference on Pattern Languages of Programs*.

[10] https://en.wikipedia.org/wiki/Analytic_hierarchy_process

[11] http://www.site.uottawa.ca/~ssome/publis/Methodology_Frameworks_Selection.pdf

[12] B. Sena, L. Garces, A. P. Allian and E. Yumi Nakagawa. 2018. Investigating the Applicability of Architectural Patterns in Big data Systems. *Pattern Languages of Programs (PLoP)*, Portland, Oregon, USA.

[13] R. E. Johnson. 1997. How frameworks compare to other object-oriented reuse techniques. Communications of the ACM, 40(10), 39-42.

[14] A. Aguiar, and G. David. 2011. Patterns for effectively documenting frameworks. *Transactions on Pattern Languages of Programming II*, 79-124, Springer.

[15] K. Beck and R. Johnson. 1994. Patterns Generate Architecture. ECOOP '94 Proceedings of the 8th European Conference on Object-Oriented Programming, pp 139-149, London, UK.

[16] Mehta, R., Ruiz-López, T., Chung, L., & Noguera, M., "Selecting among Alternatives using Dependencies: An NFR approach", *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, New York, NY, USA, pp 1292-1297, (2013).

[17] Bastos, L.R.D., & Castro, J.F.B., "Systematic Integration Between Requirements and Architecture", *Software Engineering for Multi-Agent Systems III*, pp 85-103, Volume 3390 of the series Lecture Notes in Computer Science, (2005).

[18] Zhu, M.X., Luo, X.X., Chen, X.H., & Wu, D.D., "A non-functional requirements tradeoff model in Trustworthy Software", *Information Sciences 191*, pp 61 – 75, (2012).

[19] Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L., & Yu, E., "Evaluating Goal Models within the Goal-oriented Requirement Language", *International Journal of Intelligent Systems*, pp 841-877, (August 2010).

[20] Ong, H., Weiss, M., & Araujo, I., "Rewriting a Pattern Language to Make it More Expressive", 2003.

[21] https://en.wikipedia.org/wiki/Group_decision-making

[22] N. Harrison. 2011. Improving quality attributes of software systems through software architecture patterns.

[23] S. Bode and M. Riebisch. Impact Evaluation for Quality-Oriented Architectural Decisions Regarding Evolvability. *European Conference on Software Architecture ECSA*, 2010.

[24] A. Alebrahim, S. Fassbender, M. Filipczyk, M. Goedicke, and M. Heisel. 2015. Towards a Reliable Mapping between Performance and Security Tactics, and Architectural Patterns. *EuroPLoP '15 Proceedings of the 20th European Conference on Pattern Languages of Programsv*, Article No. 39, 43 pages.

[25] G. Me, C. Calero, and P. Lago. Architectural patterns and quality attributes interaction. *2016 Qualitative Reasoning about Software Architectures (QRASA)*.

[26] N. Harrison and P. Avgeriou. 2010. Implementing Reliability: The Interaction of Requirements, Tactics and Architecture Patterns. *Architecting Dependable Systems VII* pp 97-122.

[27] M. Kassab, G. El-Boussaidi, and H. Mili. 2011. A Quantitative Evaluation of the Impact of Architectural Patterns on Quality Requirements. *Software Engineering Research,Management and Applications 2011* pp 173-184, Pp 173-184.

[28] M. Kassab and G. El-Boussaidi. 2013. Towards Quantifying Quality, Tactics and Architectural Patterns Relations. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE.

[29] https://storm.apache.org

[30] Apache Metron, metron.apache.org

[31] Apache Flink, flink.apache.org

[32] https://spark.apache.org

[33] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, E. M. Nguifo, "An experimental survey on big data frameworks", *Future Generation Computer Systems*, 546-564, 2018.

[34] https://gradle.org/maven-vs-gradle/

[35] L. Garces, B. Sena, and E. Y. Nakagawa, "Towards an architectural patterns language for System-of-Systems", HILLSIDE Proc. Of Conf. on Pattern Lang. of Prog. V (October 2019), 24 pages.